The Architecture of the MCAO RTC for the Daniel K. Inouye Solar Telescope

and

Matilda - a fast MVM library for AO RTC

Dirk Schmidt, Andrew Beard
National Solar Observatory, Boulder (Colorado), USA

ESO RTC4AO Workshop
November 2023
The Daniel K. Inouye Solar Telescope

- 4-meter off-axis Gregory design (no spider)
- Science commissioning phase since 2022
- Classical Adaptive Optics with 1600 actuators
  - FPGA+CPU hybrid, 1970 fps
- MCAO / GLAO upgrade in fabrication
- GLAO deployment (new RTC + new wavefront sensor) by end of 2024, additional DMs later
MCAO Upgrade: new WFS system, 2 additional DMs, new RTC

- 9 identical wavefront sensors
- variable field of view span
  MCAO: 48", 60"; GLAO: 2', ~2.8'
- 9.3 cm subapertures, 10" each, 20x20 px

11817 subapertures and 3454 actuators total

DM$_{4 \text{ km}}$
- 1600 actuators
- Final polishing

DM$_{11 \text{ km}}$
- 252 actuators
- FDR pending
• Infiniband HDR 200 Gb/s network

• 9\times Supermicro A+ 1114S-WTRT ("1of9", …, "9of9", "The Nine")
  • 1\times AMD Epyc 7742 CPU (64 cores, 2.25 GHz)
  • 1\times Euresys Coaxlink Octo frame grabber (currently)
  • 1\times Kaya Komodo II CLHS frame grabber (in final system)
  • 1\times CoaXPress camera, 864\times 860, 8 bit, 2 kHz (currently)
  • 1\times CameraLink HS camera, 1200\times 1200, 12 bit, 2 kHz (in final system)
  • 1\times Mellanox ConnectX-6, dual-port Infiniband adapter

• 1\times Gigabyte R282-Z91 ("unimatrix01")
  • 2\times AMD Epyc 7742 CPUs (128 cores, 2.25 GHz)
  • 2\times Mellanox ConnectX-6, dual-port Infiniband adapter
  • 1\times 10 Gb/s adapter for DM commands
  • 1\times 8K Nvidia graphics adapter for GUI
Real-time Control Loop Cycle

1. Exposure
2. Camera read out
3. Correlations, MVM shifts to modes
4. Check for timing quirks
5. Send modes to unimatrix01 via RDMA (3454)
GUI on unimatrix01 (6000 x 3400 pixels)
camera images
correlation maps
RTC Software is enhanced version KAOS Evo 2

- KAOS Evo 2 originally developed for Gregor at Kiepenheuer-Institute for Solarphysics (Thomas Berkefeld & I)
- Branched off in 2013 at National Solar Observatory for Big Bear Solar Observatory
  "Clear" (MCAO), AO308 II (CAO), GLAO, prominence AO
- Linux application, runs completely on x86-64 CPUs
- Complete: Control Loop and all calibrations integrated, nothing is done via external software
- GUI: designed for the operator
- Super flexible: almost everything is in config files, few performance sensitive parameters are compile time constants.
- Supports multiple SH-WFSs (cameras), each WFS can have multiple guide-regions

- Plugin-based hardware support
  - Euresys, EDT, Kaya, Active Silicon frame grabbers
  - various camera models (Mikrotron, Adimec, Photonfocus, First Light Imaging, Optronis, Dalsa)
  - Riptide Realtime DM Xinetics interface (+ obsolete in-house solutions)

- Low single-digit RMS jitter (on well tuned systems, < 2 µs on any of The Nine)
RTC is enhanced version KAOS Evo 2

- divided into real-time back-end and GUI (Qt),
- uses both POSIX Shared Memory and threads
- “Real-time” application
  - Tuned hardware and operating system
  - Control loop threads use SCHED_FIFO with priority 94
  - Threads pinned on dedicated CPU cores
  - Threads take 100% of each assigned core and never yields the CPU (no mutexes, usleep(), etc.)
  - RMS jitter < 2 µs (per host)
Low-latency tuning and customization for DKIST MCAO RTC
Enhancing hardware determinism and isolating CPU cores

- Consult tuning guides for HPC and low-latency applications by hardware vendors, e.g.
  - AMD “Workload Tuning Guide for AMD EPYC 7002 Series Processor Based Servers”
  - AMD “Performance Tuning Guidelines for Low Latency Response on AMD EPYC-Based Servers”
  - AMD “Performance Tuning Guidelines for Low Latency Response on AMD EPYC™ 7002 Series Processor Based Servers”
  - AMD “HPC Tuning Guide for AMD EPYC Processors”

- Custom BIOS from Supermicro on The Nine
  - Exposures features not found in default BIOS

- BIOS settings synchronized across servers via Redfish
  - Disabled: ASPMSupport, CorePerformanceBoost, DRAMScrubTime, HighPrecisionEventTimer, IOMMU, SMEE, SMTControl, SR-IOVSupport, SVMMode, TSME, WatchDogFunction
  - APBDIS=1, DeterminismSlider=Performance, EfficiencyModeEn=Auto, FixedSOCPstate=P0, LocalAPICMode=xAPIC
Low-latency tuning and customization for DKIST MCAO RTC
Linux tweaking, isolating CPU cores and reducing operating system jitter

- Debian Linux 11 with stock “linux-image-amd64” kernel
- unimatrix01 runs KDE desktop on 8k display (primary user interface)
- Follow “Reducing OS jitter due to per-cpu kthreads” guide
- LD_PRELOAD a custom library to prevent unwanted programs from setting their CPU affinities to isolated cores
  - overwrites sched_setaffinity() and pthread_setaffinity_np()
  - keeps “intrusive” programs which ignore preset affinities from freezing when AO RTC software is running
- All root file systems are clones from 1of9
  - Root file systems are read-only to prevent “accidental” and unrecorded changes
  - /home and /var on separate partitions, no swap partition (/var is read+write and contains symlinks for adjtime, resolv.conf, blkid.tab, etc.)
Low-latency tuning and customization for DKIST MCAO RTC

Linux tweaking, isolating CPU cores and reducing operating system jitter

# kernel command line
acpi_irq_nobalance apparmor=0 audit=0 clocksource=tsc cpufreq.default_governor=performance cpuuidle.off=1 idle=poll irqaffinity=0-7 isolcpus=8-63 mce=off mitigations=off nmi_watchdog=0 nohz=on nohz_full=8-63 noirqbalance nosoftlockup=0
nowatchdog processor.max_cstate=0 rcu_nocbs=8-63 rcu_nocb_poll=8-63 selinux=0 skew_tick=1 tsc=reliable

# /etc/systemd/system/cpus-offline-online.service
[Unit]
Description=Disable and re-enable CPUs 8-63
After=network.target sshd.service
[Service]
Type=oneshot
ExecStart=/bin/sh -c 'sleep 60 && chcpu --disable 8-63 && chcpu --enable 8-63'
RemainAfterExit=no
StandardOutput=journal
[Install]
WantedBy=default.target

# /etc/sysctl.d/kaos.conf
kernel.nmi_watchdog = 0
kernel.numa_balancing = 0
kernel.hung_task_timeout_secs = 0
kernel.sched_rt_runtime_us = -1
kernel.randomize_va_space = 0
vm.swappiness = 0
kernel.sched_tunable_scaling = 0
kernel.timer_migration = 0
vm.stat_interval = 10

# /etc/security/limits.d/kaos.conf
@aousers    hard    memlock         40000000
@aousers    soft    memlock         40000000
@aousers    -       rtprio          99
@aousers    -       nice            -19
KAOS Evo 2 design principles

• "C-style" C++

• Data layout designed (refactored) for SIMD architectures
  • "structures of arrays" where vectorization and contiguous data access is beneficial, e.g. slope, mode and actuator data
  • "arrays of structures" where not beneficial and to prevent "false sharing" of cache lines between several threads, e.g. slope, other subaperture data
  • all arrays forcefully aligned on cache line boundaries
    • not NUMA optimized (yet?), all data is local to first real-time core

• Automatic and mandated vectorization using #pragma omp simd
  • Analysis of vectorization reports, benchmarking and profiling (Intel ICC, Advisor, V-Tune, Compiler Explorer) to verify and improve
  • Intel ICC usually creates faster code than GCC, Clang (AOCC, Intel ICX) even on AMD

• Lock-free
  • no simultaneous reads and writes to the same variable by different threads
  • simultaneous writes to the same cache line are rare, too

• Thread synchronization based on busy-looping (100% CPU use)
  • only three thread synchronization barriers per loop cycle (compute shifts, modes, and actuators)
  • one atomic integer per thread, value specifies which task to execute or finished state
  • lowest overhead on AMD Epyc appears with 4 integers per cache line despite false sharing (2 µs for 64 threads on AMD 7742)

• Aiming for "branchless" code where it’s easy and reasonable
Clusterization of KAOS Evo 2

- Originally shared memory application

- Extending to cluster application
  - RDMA over Infiniband (libibverbs, librdmacm)
    - tested various write/read mechanisms to identify lowest latency approach
      - sender: RDMA_WRITE operation
      - receiver: polling raw memory (busy-looping)
    - latency: ~10 µs to send 4000 32-bit numbers from all Nine to unimatrix01

- RDMA layer complete, basic control loop running at 2000 Hz
  - no correlation reference updating, frequency filtering, etc. yet

- Command layer (rpclib) almost complete

- GUI working
Simulation-based Testing of RTC with Blur

Blur: NSO’s solar AO simulator

- receives:
  - actuator commands
    (via ethernet)

- models:
  - deformable mirrors,
    turbulent and windy atmosphere,
    imaging of extended source through atmosphere,
    imaging noise

- outputs:
  - simulated wavefront sensor camera images
    (via CoaXPress / CameraLink simulators, shared memory, or ethernet)

- plugs directly into RTC instead of cameras

- Full DKIST MCAO at 10 fps on dual AMD 7742

- 1× Supermicro AS4125GS-TNRT (“chaotica”)
  - 2× AMD Epyc 9654 CPUs (192 cores, 2.4 GHz)
  - 9× Kaya Chameleon II CoaXPress simulator
9 PCIe slots, 192 Zen 4 cores....

How many wavefront sensors can KAOS Evo 2 process on chaotica?
9 PCIe slots, 192 Zen 4 cores....

How many wavefront sensors can KAOS Evo 2 process on chaotica?

Nine!
(at 1860+ fps using 160 cores)

Now considering and evaluating switching gears and run RTC on 1 or 2 computers only. Final MVM on GPU or unimatrix01?
Throtteling in AMD’s 2nd Gen Epyc CPUs

MVM slows down when busy waiting for next MVM

AMD 7742 CPUs slow down
- All power savings and frequency scalings deactivated in BIOS and Linux (system tuned for low latency)
- Only seen on AMD Zen 2 CPUs (not on Intel Broadwell, Skylake, Cascade Lake, or AMD Zen 4)
- Seems to happen inside the CPU
- CPU slows down whenever deviated from full-size MVM for a short while

“Solutions”:
- Keep MVM going at full size, disregard output
  - Costs additional latency for waiting until dummy MVM has been completed
- Avoid Zen 2 (can anybody test on Zen 1 and Zen 3?)
# Matilda - a fast, low-jitter MVM library for AO RTC

**Designed and optimized for repeated MVMs with constant matrix on many-core CPUs**

<table>
<thead>
<tr>
<th>rows x columns</th>
<th>MVM timing</th>
<th>CPUs</th>
<th>threads</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024 x 1024</td>
<td>~4 µs</td>
<td>AMD Epyc 7742</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>~6 µs</td>
<td>AMD Epyc 7742</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>1280 x 3500</td>
<td>~10 µs</td>
<td>AMD Epyc 9654</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>~13 µs</td>
<td>AMD Epyc 7742</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>2048 x 2048</td>
<td>~9 µs</td>
<td>AMD Epyc 7742</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>~10 µs</td>
<td>Intel Xeon Gold 6254</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>~25 µs</td>
<td>Intel Xeon E5-2698 v4</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>3584 x 3456</td>
<td>~15 µs</td>
<td>AMD Epyc 9654</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>(DKIST MCAO,</td>
<td>~24 µs</td>
<td>AMD Epyc 7742</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>modes to</td>
<td>~27 µs</td>
<td>Intel Xeon Gold 6254</td>
<td>32</td>
<td>F16C</td>
</tr>
<tr>
<td>actuators)</td>
<td>~67 µs</td>
<td>Intel Xeon E5-2698 v4</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>~107 µs</td>
<td>Intel Xeon Gold 6254</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>4096 x 4096</td>
<td>~17 µs</td>
<td>AMD Epyc 9654</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td></td>
<td>~32 µs</td>
<td>AMD Epyc 7742</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>~40 µs</td>
<td>Intel Xeon Gold 6254</td>
<td>32</td>
<td>F16C</td>
</tr>
<tr>
<td></td>
<td>~95 µs</td>
<td>Intel Xeon E5-2698 v4</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>~200 µs</td>
<td>Intel Xeon Gold 6254</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>4608 x 4096</td>
<td>~25 µs</td>
<td>AMD Epyc 7742</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td></td>
<td>~38 µs</td>
<td>AMD Epyc 7742</td>
<td>64</td>
<td>Intel MKL</td>
</tr>
<tr>
<td></td>
<td>~105 µs</td>
<td>AMD Epyc 9654</td>
<td>96</td>
<td>Intel MKL</td>
</tr>
<tr>
<td>8192 x 8192</td>
<td>~55 µs</td>
<td>AMD Epyc 9654</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>12800 x 3500</td>
<td>~24 µs</td>
<td>AMD Epyc 9654</td>
<td>160</td>
<td>F16C</td>
</tr>
<tr>
<td></td>
<td>~31 µs</td>
<td>AMD Epyc 9654</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>19200 x 4092</td>
<td>~31 µs</td>
<td>AMD Epyc 9654</td>
<td>150</td>
<td>F16C</td>
</tr>
<tr>
<td></td>
<td>~52 µs</td>
<td>AMD Epyc 9654</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>33280 x 8192</td>
<td>~105 µs</td>
<td>AMD Epyc 9654</td>
<td>160</td>
<td>F16C</td>
</tr>
<tr>
<td></td>
<td>~820 µs</td>
<td>AMD Epyc 9654</td>
<td>160</td>
<td></td>
</tr>
</tbody>
</table>

- Matilda is not a general purpose BLAS gemv() interface
  MVM is usually memory bandwidth bound, but BLAS (=contiguous block of matrix data) is not bandwidth optimal on modern many-core CPUs with NUMA caches and RAM

- Matilda: First optimize data layout, then optimize the computation
  Create an “mvm_plan” (memory, threads) for one particular matrix in the beginning, then execute the plan for to compute the MVM.

- MVM plan is executed on dedicated, reserved CPU cores
  Matrix data cannot be evicted from caches

- Parallelized using its own thread pool designed for low latency/jitter
  - Threads are pinned to fixed CPUs and scheduling priorities can be elevated
  - Threads remain active when MVM has finished

- Explicit, hand-tuned vectorization via AVX2 and AVX-512 intrinsics

- Supports F16C 16-bit floating point format for matrix data storage to minimize cache use (MVM computed in 32-bit precision)

https://github.com/nsomatilda/Matilda
Matilda - a fast MVM library for AO RTC

Matrix data partitioning and layout

- Matrix data partitioned for threads
  - each thread gets a horizontal slice with multiples-of-8 rows
  - number depends on selected kernel and thread number
- Slice data transposed to column-major order
- Copied into core-local memory pages
  - aligned on cache line boundary
- Matrix data is not contiguous anymore, fundamental difference to BLAS.
  - High setup overhead upfront
  - More efficient memory access later

Matrix stored in row-major format

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>50</td>
</tr>
</tbody>
</table>

Matrix reordered in column-major format

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>21</td>
<td>31</td>
<td>41</td>
<td>51</td>
<td>61</td>
<td>71</td>
<td>81</td>
<td>91</td>
</tr>
<tr>
<td>12</td>
<td>22</td>
<td>32</td>
<td>42</td>
<td>52</td>
<td>62</td>
<td>72</td>
<td>82</td>
<td>92</td>
<td>10</td>
</tr>
<tr>
<td>13</td>
<td>23</td>
<td>33</td>
<td>43</td>
<td>53</td>
<td>63</td>
<td>73</td>
<td>83</td>
<td>93</td>
<td>11</td>
</tr>
<tr>
<td>14</td>
<td>24</td>
<td>34</td>
<td>44</td>
<td>54</td>
<td>64</td>
<td>74</td>
<td>84</td>
<td>94</td>
<td>12</td>
</tr>
<tr>
<td>15</td>
<td>25</td>
<td>35</td>
<td>45</td>
<td>55</td>
<td>65</td>
<td>75</td>
<td>85</td>
<td>95</td>
<td>13</td>
</tr>
<tr>
<td>16</td>
<td>26</td>
<td>36</td>
<td>46</td>
<td>56</td>
<td>66</td>
<td>76</td>
<td>86</td>
<td>96</td>
<td>14</td>
</tr>
<tr>
<td>17</td>
<td>27</td>
<td>37</td>
<td>47</td>
<td>57</td>
<td>67</td>
<td>77</td>
<td>87</td>
<td>97</td>
<td>15</td>
</tr>
<tr>
<td>18</td>
<td>28</td>
<td>38</td>
<td>48</td>
<td>58</td>
<td>68</td>
<td>78</td>
<td>88</td>
<td>98</td>
<td>16</td>
</tr>
<tr>
<td>19</td>
<td>29</td>
<td>39</td>
<td>49</td>
<td>59</td>
<td>69</td>
<td>79</td>
<td>89</td>
<td>99</td>
<td>17</td>
</tr>
<tr>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
<td>18</td>
</tr>
</tbody>
</table>

SIMD computation on column-major buffer

<table>
<thead>
<tr>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
<th>V</th>
<th>VI</th>
<th>VII</th>
<th>VIII</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
</tr>
<tr>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
</tr>
<tr>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
</tr>
<tr>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
</tr>
<tr>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
<td>80</td>
</tr>
<tr>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
</tr>
<tr>
<td>89</td>
<td>90</td>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
<td>96</td>
</tr>
<tr>
<td>97</td>
<td>98</td>
<td>99</td>
<td>100</td>
<td>101</td>
<td>102</td>
<td>103</td>
<td>104</td>
</tr>
</tbody>
</table>

- contiguous loading of matrix elements
- vector elements re-used (only loaded once)
- only SIMD FMA operations used

24 SIMD FMA operations only

- does not need horizontal reduction
- (very simple access pattern)