OPTIMIZING EPICS FOR MULTI-CORE ARCHITECTURES*

Ralph Lange, Helmholtz-Zentrum Berlin / BESSY II, 12489 Berlin, Germany
Franck Di Maio, ITER Organization, St. Paul lez Durance, France

Abstract

EPICS is a widely used software framework for real-time controls in large facilities, accelerators and telescopes. Its multithreaded IOC (Input Output Controller) Core software has been developed on traditional single-core CPUs. The ITER project will use modern multi-core CPUs, running the RHEL Linux operating system in its MRG-R real-time variant. An analysis of the thread handling in IOC Core shows different options for improving the performance and real-time behavior, which are discussed and evaluated. The implementation is split between improvements inside EPICS Base, which have been merged back into the main distribution, and a support module that makes full use of these new features. This paper describes design and implementation aspects, and presents results as well as lessons learned.

INTRODUCTION

The ITER project hardware platform for fast controllers (FC) will rely on modern industrial PCs with fast multi-core Intel CPUs, running the MRG-R real-time version of the Red Hat Enterprise Linux operating system. External vendors preparing fast controls for their subsystems are concerned about the real-time properties of their fast control loops running inside or outside the EPICS IOC on the FC, when the IOC is running on the same machine.

Thread handling within the EPICS IOC has been designed with single-core processors in mind. Allowing to fine-tune the system by dedicating specific cores or core sets to either specific threads of the IOC or the external control loop requires extensions of the EPICS Base software and addition of multi-core and OS specific functions using those extensions.

We describe the thread handling inside the EPICS IOC, identify areas for optimization, present design and implementation of the MultiCore Utilities package and the changes to EPICS Base, show first results, and highlight ways for further improvement.

THREAD HANDLING IN EPICS

IOC Threads

From the very beginning of the EPICS project [1], its IOC application has been multithreaded. Fig. 1 shows the threads running on an EPICS 3.15 IOC, without any sequencer state machines or Channel Access (CA) communication being active.

```
epics> epicsThreadShowAll
            NAME     EPICS ID   LWP ID   OSIPRI  OSSPRI  STATE
          _main_    0x11e21f0    32488      0       0       OK
          errlog    0x11ec090    32489     10       0       OK
          taskwd    0x11f2260    32490     10       0       OK
      timerQueue    0x127f200    32491     70       0       OK
           cbLow    0x127f4e0    32492     59       0       OK
        cbMedium    0x1280620    32493     64       0       OK
          cbHigh    0x1280930    32494     71       0       OK
        dbCaLink    0x1280de0    32495     50       0       OK
        scanOnce    0x127b0c0    32496     70       0       OK
         scan-10    0x127bcd0    32497     60       0       OK
          scan-5    0x127bf50    32498     61       0       OK
          scan-2    0x127c1d0    32499     62       0       OK
          scan-1    0x127c450    32500     63       0       OK
        scan-0.5    0x127c6d0    32501     64       0       OK
        scan-0.2    0x128c3e0    32502     65       0       OK
        scan-0.1    0x128c620    32503     66       0       OK
         CAS-TCP    0x128ce30    32504     18       0       OK
      CAS-beacon    0x128d140    32505     17       0       OK
         CAS-UDP    0x128d440    32506     16       0       OK
       CAC-event    0x11e7ab0    32507     51       0       OK
epics>
```

Figure 1: Threads on an empty IOC.

These threads belong to different functional groups.

Scan Threads. Periodical scanning of records is performed by one thread for each defined scan period, that keeps a list of the records that have their SCAN field set to that period. Whenever a scan thread wakes up, it processes the records in its list, then sleeps until it is due for the next period. The scan once thread handles one-time processing of records, working off a FIFO queue of records instead of a list.

The assigned thread priorities range between 60 and 70, with scan threads for smaller periods being assigned higher priorities, so that in a real-time environment faster scan threads will preempt the slower ones.

Callback Threads. Three general purpose callback threads handle the processing of I/O triggered records. Like the scan once thread they work off queues, and interrupt handlers request processing of records by an API call that pushes the request on the appropriate queue.

These threads are mapped to the three available record priorities, and are assigned the priorities 59 (LOW, below scan threads), 64 (MEDIUM, in the middle of the scan thread range), and 71 (HIGH, above scan threads).

Channel Access Threads. An IOC without external Channel Access clients runs five CA threads. The CAS-UDP thread (priority 16) handles incoming naming requests. The CAS-beacon thread (priority 17) sends UDP alive messages to clients. The CAS-TCP thread (priority 18) listens for incoming TCP connections, and spawns off the individual communication threads. The dbCaLink thread (priority 50) handles the CA client-side connections that originate from links in the local database. The CAC-event thread (priority 51) handles the local CA

*Work supported in part by German Bundesministerium für Bildung und Forschung and Land Berlin.

ISBN 978-3-95450-139-7

updates that have to be sent to fields inside the local database.

For every new remote CA client that connects and every new outgoing CA connection, one pair of threads is created, servicing the TCP circuit that connects to the peer. The CAS-event thread (priority 19-39) handles sending out the CA updates to the client, while the CAS-client thread (priority of CAS-event + 1) handles the incoming messages. The CAC-TCP-recv thread (priority 49) handles incoming messages from the remote server, while the CAC-TCP-send thread (priority 51) sends outgoing messages.

Miscellaneous EPICS Threads. The _main_ thread runs the IOC shell. The errlog thread (priority 10) forwards error messages to configured listeners, locally or across the network. The taskwd thread (priority 10) monitors the status of threads that have registered with it. timerQueue threads are started by the EPICS timer facility, whenever the user creates a timer queue. The Access Security layer starts the asCaTask thread (priority 57) if it needs to connect through CA to determine access right conditions.

Sequencer State Sets. Each sequencer state machine is running in its own thread (priority 50), the sequencer also starts an auxiliary thread seqAux (priority 51).

Other Threads. Other applications, drivers, and the operating system itself may start additional threads.

Thread Scheduling and Policy

On dedicated real-time operating systems, EPICS is using the system scheduler's policy. E.g., vxWorks and RTEMS use combinations of strict priority-based scheduling and round-robin policy.

On Linux, without any additional configuration and fine-tuning, the default Linux scheduler will control scheduling of threads and their distribution over the available CPUs. In this case, the priorities are ignored.

The scheduler used in the Red Hat Enterprise Linux (RHEL) version 6 kernel (2.6.32) is the Completely Fair Scheduler (CFS) by Ingo Molnár [2]. The group scheduling improvement added later to the 2.6.38 kernel [3] does not affect real-time systems, as it addresses only machines in the desktop and workstation classes.

The “fair queuing” algorithm that CFS implements ensures that threads with little activity and threads with a lot of activity share the available CPU resources in a fair way. For the regular EPICS threads inside an IOC without real-time duties, this scheduler will suffice.

When priority scheduling is enabled, all EPICS threads will be started using SCHED_FIFO scheduling policy (see chapter 4.2), and the priorities will be used.

There is no EPICS API for setting or changing the scheduling policy, thread priorities for EPICS threads are hard-coded.

EPICS ON MULTI-CORE (SMP) SYSTEMS

The EPICS IOC has been developed for single-core systems. With the introduction of the 3.14 branch of EPICS Base that allowed IOCs to be run as processes on host type systems, all necessary mechanisms were added to safely run the IOC on multi-core (SMP) architectures. Many installations are running large numbers of host-based IOCs in production, usually on Intel- or Sun-based hardware, often in virtualized environments.

As multi-core CPUs have been entering the market for embedded and real-time hardware only slowly, most EPICS installations still run their real-time systems on single-core CPUs (mostly PPC architectures using VME or Compact-PCI form factor), and no extensive optimizations for running real-time IOCs on SMP systems have been made.

Dedicated Cores

Multi-core real-time systems like the ITER Fast Controller might want to guarantee execution time to specific driver threads by reserving cores exclusively. To achieve this, the available set of cores may be split between the EPICS IOC and specific real-time threads by setting the thread CPU affinity to disjunctive subsets, and routing interrupts to the appropriate cores by setting the interrupt CPU affinity.

The EPICS IOC may run on any number of cores.

There is no EPICS API for setting or changing CPU affinity for threads or interrupts.

Possible IOC Thread Improvements

Scan Threads. The periodic scan threads are spread onto the available CPU cores by the Linux scheduler. They will run in parallel, should they be due at the same time. The lock sets used by EPICS (see chapter 5.5 “Database Locking” in [4]) provide proper locking of records across CPUs. Splitting the work of a single scan thread over multiple cores would improve the throughput of processing the records on one scan period list, without affecting the real-time behavior.

On the other hand, periodic scans honor the PHAS field of EPICS records. I.e., for any record with a given PHAS value it is guaranteed that before it starts processing, all records with a lower PHAS value in the same scan period have finished processing. The current implementation achieves this by having exactly one thread processing a list of records sorted by PHAS value. To ensure PHAS order synchronization across multiple tasks, the end of processing would have to be tracked and communicated between parallel tasks, a major change.

Callback Threads. The callback threads can also be run on different CPU cores by the scheduler, so that

ISBN 978-3-95450-139-7

callback record processing for I/O scanned records of different priority can be executed in parallel. Running multiple callback threads for the same priority in parallel on different CPUs would immediately shorten the average use of the callback queue and noticeably reduce interrupt-to-record-processing latency, which is a key figure for EPICS real-time behavior. Parallel callback threads would also greatly improve deterministic behavior of EPICS, which is worsened by using queues between interrupts and record processing. (See the discussion in [5].)

Channel Access Threads. The Channel Access communication works asynchronously, with low priority, and queues or buffers data updates at multiple levels. It does not affect the real-time behavior of an IOC. The default scheduler should do an excellent job for the CA threads.

Sequencer State Sets. A sequencer state set implements a finite state machine, that switches between a fixed set of states and transitions. A single thread is an excellent representation for a state machine, and no gains can be expected through parallelization.

Other Threads. The effects and side-effects of parallelization and scheduling for other threads highly depend on their specific functionality.

As mentioned above, real-time threads that are part of a low-lever driver or a fast feedback loop might have a good chance of improving their deterministic behavior when run on a dedicated CPU.

PARALLEL CALLBACK THREADS

Running parallel callback threads adds multiple consumers to each of the three priority-mapped queues. To allow an efficient implementation of a thread-safe queue, a spin lock API was added to the Operating System Independent (OSI) layer of EPICS, with implementations for all supported platforms and architectures. The spin locks allowed for a thread-safe variant of the generic queues, and parallel callback threads were added to EPICS Base.

These changes have been proposed for merging into EPICS Base 3.15.

EPICS MULTI-CORE UTILITIES

An EPICS Multi-Core Utilities library was created [6], that contains tools to allow tweaking of real-time parameters for EPICS IOC threads running on multi-core processors under the Linux operating system.

These tools are intended to set up multi-core IOCs for fast controllers, by:

- Confining either parts or the complete EPICS IOC onto a subset of the available cores, allowing hard real-time applications and threads to run on dedicated cores.
- Changing priorities of callback, driver or communication threads with respect to database processing.
- Selecting real-time scheduling policy (FIFO or Round-Robin) for selected threads.
- Locking the IOC process virtual memory into RAM to avoid swapping.

Rule-Based Thread Properties

This module allows user-specified rules to modify real-time properties of EPICS IOC threads:

- *Scheduling policy*: Scheduling mechanism used for the thread. When POSIX scheduling is enabled, the default mechanism is FIFO, but OTHER and Round Robin are also supported.
- *Scheduling priority:* EPICS priority value that gets converted to the OS real-time priority schema. Absolute and relative values are supported.
- *CPU Affinity:* Set of CPUs that the thread is allowed to run on.

Rules are read from system or user level rules files, each specifying a regular expression and operations on these properties. The operations are executed when the regular expression matches the EPICS thread name.

Commands for the EPICS iocShell allow directly manipulating the properties of any existing thread, and configuring the active set of thread rules.

Advanced Thread Show Routines

The existing EPICS thread show routines have been extended to show scheduling policy and CPU affinity in addition to the usual output.

Memory Locking

Functions (also available from the iocShell) are provided that allow locking the process memory into RAM to make sure no page faults occur, which would introduce unpredictable interruptions and latency.

Implementation

The implementation was split in two parts:

All generic changes were stripped down for minimal impact and integrated into the EPICS Base 3.15 libraries. Most of these changes consist of adding hooks that allow user code to be called in certain situations, e.g., whenever a thread is created in the IOC.

All changes specific to multi-core CPUs or the operating system were added to the Multi-Core Utilities library, that makes full use of the new hook facilities in EPICS Base.

The changes in EPICS Base have been folded back into the main line development branch, the Multi-Core Utilities were published in the EPICS Applications project on SourceForge [6].

ISBN 978-3-95450-139-7

FIRST RESULTS

A sample user level application (based on the C++ asynPortDriver class described in [7]) has been created, featuring a fast control loop inside an IOC. The code uses a timing card and an I/O card from the ITER FC hardware catalog to sample a frequency generator signal and produce marker signals for time measurements.

Software time measurements are taken and fed into histogram records to show their distribution over many iterations of the control loop. The hardware marker signals are used for an oscilloscope-based validation of the software time measurements.

While many properties of the setup still have to be tested and their influence quantified, Fig. 2 shows the complete span of achievable improvements. The histogram above shows the overall control loop time without any modification to the thread system, the histogram below shows the same timing with separation of the control loop onto a dedicated core, and after increasing its scheduling priority.

The test is run on a dual-core CPU system of the "workhorse" FC type.

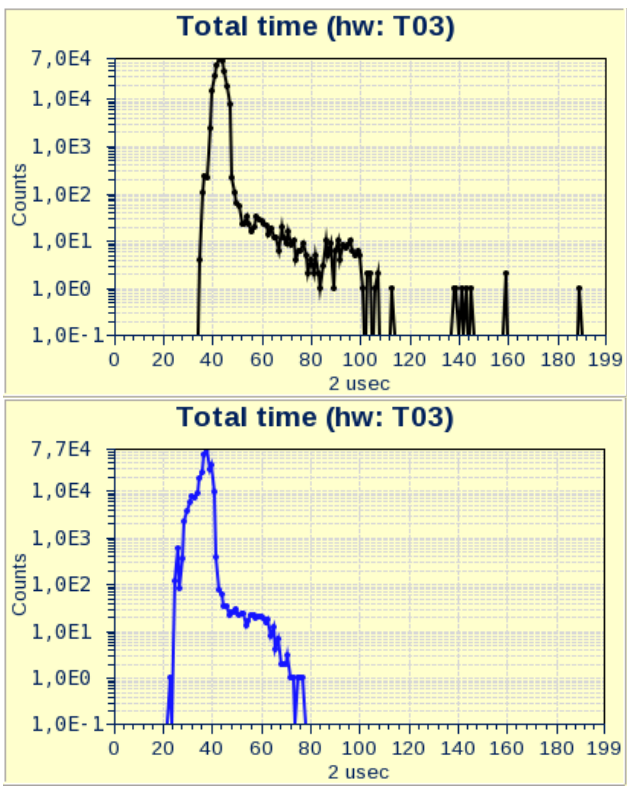

Figure 2: Total Control Loop Time Measured Without and With Core Affinity and Priority Tuning.

FURTHER IMPROVEMENTS

The combination of parallel callback threads and thread resource allocation already greatly reduces the interrupt-to-record latency and makes better use of the computing power of multi-core CPUs. Still, the callback system is a general purpose facility, and other parts of the IOC software can offload work to the callback threads, which may affect the real-time behavior.

The callback threads or thread sets map to the three available record priorities. Adding user-defined options to that priority enumeration would allow drivers to define dedicated "private" callback threads with their own separate queue instances, that are only used by that specific driver.

The driver could also specify the queue length to control maximum latency and queue/cache preference, and a special queue length could even process the records directly from the driver thread context, eliminating all queues and thread switches, allowing almost fully deterministic behavior for control loops implemented as EPICS records.

CONCLUSIONS

The implemented EPICS extensions and utilities show that the real-time behavior of the EPICS IOC on multi-core CPUs can be optimized by fine-tuning the real-time properties and thread resource allocation.

Other options have been identified, that would further extend the IOC into being an engine for deterministic fast real-time control, eventually meeting the requirements for applications like plasma control, while retaining the full flexibility and versatility of the IOC and connecting such applications to the rich tool set of the EPICS framework.

ACKNOWLEDGMENT

We would like to thank the ITER CODAC team and the EPICS Base Developers for their cooperation, help, and fruitful discussions.

REFERENCES

[1] Experimental Physics and Industrial Control System, http://www.aps.anl.gov/epics.

[2] I. Molnár, "Modular Scheduler Core and Completely Fair Scheduler [CFS]", Linux-Kernel mailing list, 2007, http://lwn.net/Articles/230501.

[3] M. Galbraith, "sched: automated per tty task groups", Linux-Kernel mailing list, 2010, http://marc.info/?l=linux-kernel&m=128978361700898.

[4] M. Kraimer et al., "EPICS Application Developer's Guide", edition for EPICS Base Release 3.14.12, http://www.aps.anl.gov/epics/base/R3-14/12-docs/AppDevGuide.pdf.

[5] A. Barbalace et al., "Comparative Analysis of EPICS IOC and MARTe for the Development of a Hard Real-Time Control Application", ICALEPCS2011, Grenoble, October 2011, WEPMN036, pp. 961-964 (2011), http://www.JACoW.org.

[6] R. Lange, "EPICS Multi-Core Utilities" documentation, http://epics.sf.net/mcoreutils.

[7] M. Rivers, "asynPortDriver – C++ Base Class for Asyn Port Drivers", Chicago, February 2009, http://www.aps.anl.gov/epics/modules/soft/asyn/R4-12/asynPortDriver.html

ISBN 978-3-95450-139-7