Fusion Engineering and Design 89 (2014) 853–858

Contents lists available at ScienceDirect

Fusion Engineering and Design

journal homepage: www.elsevier.com/locate/fusengdes

NSTX-U Control System Upgrades

K.G. Erickson*, D.A. Gates, S.P. Gerhardt, J.E. Lawson, R. Mozulay, P. Sichta, G.J. Tchilinguirian

Princeton Plasma Physics Laboratory, PO Box 451, Princeton, NJ 08543-0451, USA

ARTICLE INFO

Article history:
Received 28 May 2013
Received in revised form 28 April 2014
Accepted 29 April 2014
Available online 21 June 2014

Keywords:
NSTX-U
PCS
Real time control
Linux
RedHawk

ABSTRACT

The National Spherical Tokamak Experiment (NSTX) is undergoing a wealth of upgrades (NSTX-U). These upgrades, especially including an elongated pulse length, require broad changes to the control system that has served NSTX well. A new fiber serial Front Panel Data Port input and output (I/O) stream will supersede the aging copper parallel version. Driver support for the new I/O and cyber security concerns require updating the operating system from Redhat Enterprise Linux (RHEL) v4 to RedHawk (based on RHEL) v6. While the basic control system continues to use the General Atomics Plasma Control System (GA PCS), the effort to forward port the entire software package to run under 64-bit Linux instead of 32-bit Linux included PCS modifications subsequently shared with GA and other PCS users. Software updates focused on three key areas: (1) code modernization through coding standards (C99/C11), (2) code portability and maintainability through use of the GA PCS code generator, and (3) support of 64-bit platforms. Central to the control system upgrade is the use of a complete real time (RT) Linux platform provided by Concurrent Computer Corporation, consisting of a computer (iHawk), an operating system and drivers (RedHawk), and RT tools (NightStar). Strong vendor support coupled with an extensive RT toolset influenced this decision. The new real-time Linux platform, I/O, and software engineering will foster enhanced capability and performance for NSTX-U plasma control.

1. Introduction

The NSTX Control System (NCS) [4,5] currently provides control capabilities to achieve the physics objectives of the National Spherical Tokamak Experiment (NSTX) [1]. The NCS runs on a real time computer with a modified Linux operating system and communicates with the outside world using the Front Panel Data Port (FPDP) protocol. The actual plasma control algorithms [4,5,9] run within the framework of the General Atomics Plasma Control System (PCS) [6–8] as a component of the overall NCS. Other NCS components implemented by NSTX convert the physics-oriented outputs of the PCS into various formats for the hardware to understand.

NSTX is currently undergoing a multi-year upgrade (NSTX-U) [2]. This upgrade targets multiple critical areas for spherical torus research, including high-beta non-inductive operation [3], plasma profile control, and boundary [3a] and divertor optimization [3b,3c], all of which require advanced control capabilities. To improve maintainability and provide enhanced support for NSTX-U operations, the NCS requires significant upgrades. Section 2 describes the new real time computer hardware. Section 3 describes the underlying FPDP transport system upgrade. Section 4 describes the required OS changes. Section 5 describes the numerous software improvements. Section 6 summarizes the upgrade activities and presents directions for future work. The enhanced performance provided by this system will enable improved plasma control, facilitating a reduced disruption rate [10] and thus enabling the critical enhanced pulse length mission of the NSTX-U facility.

2. New real time computer

To prepare for a series of control system upgrades, it proved effective to first purchase a prototype machine to test the proposed platform. This prototype closely matches the eventual purchase for NSTX-U, updated to keep pace with technology. Currently, the specifications for the new computer include 64 2.8 GHz cores across four 16-core AMD Opteron 6386 SE processors on a SuperMicro H8QG6-F mainboard with 64 GB 1866 MHz Registered ECC memory. This is a factor of 8 improvement in processing power over the previous generation [5] control computer. Also included are several PCI Express cards: a CUDA-capable video card for future GPU programming, two 4-port fiber optic SL240 FPDP cards, and a Concurrent

* Corresponding author. Tel.: +1 609 243 3064.
E-mail address: kerickso@pppl.gov (K.G. Erickson).

http://dx.doi.org/10.1016/j.fusengdes.2014.04.069

Realtime Clock and Interrupt Module (RCIM). This leaves two free slots for future use.

Local files stored on disk predominantly consist of the operating system installation and associated software packages. Any customized configuration files (i.e./etc/*) remain under revision control [18], and the entire system is backed up nightly [17]. All NSTX-U specific software, data, and related files live on a remote NFS mount. The NSTX-U configuration still utilizes a RAID, however it is primarily to insure against failures that would disrupt operations rather than to guarantee data integrity. This allows plasma discharges to continue without interruption should a hard drive fail during the run day. We can then repair or replace when there is less impact to our overall mission.

The RAID consists of two disks arranged in a RAID 1 mirror, a hot spare, and a scratch disk. A RAID 5 would of course be possible, but such sophistication serves no benefit given the primary purpose of the RAID in the NSTX-U configuration. The controller is an onboard LSI MegaRAID for increased reliability without incurring significant CPU overhead. This is technically a firmware/driver based RAID, thus optimum performance requires some software support (typically via the dmraid tool). However, testing has shown this to be less intrusive than a full software solution using the mdadm Linux software RAID utility and more cost effective than a full hardware solution.

The new computing platform fully supports version 2.0 of the Intelligent Platform Management Interface (IPMI). This is essentially a second computer running on the same motherboard as the main host computer with its own ethernet management port and direct access to certain hardware facilities. It provides a remote user the ability to perform hard resets or even low level BIOS maintenance actions that previously required physical console access. SuperMicro's IPMI implementation includes access via a web browser connecting to a web server running on the management port, as well as access via ssh and access via any standard IPMI tool.

To support online testing with live data, NSTX installed a second identical computer attached to the FPDP input stream, while the output stream remained connected solely to the primary computer. In this way, a user could exercise a new or modified algorithm on a live system and with live data but without impacting operations. This technique changed the algorithm rollout sequence to include live testing prior to final deployment on the real system. NSTX-U will further extend the concept beyond plasma control algorithm enhancements. Operating system changes, hardware changes, policy changes, and anything else about the entire operating environment will receive extensive testing in a non-intrusive way to greatly reduce the possibility of either (1) new advancements breaking the system, or (2) stagnation for fear of (1).

In tandem with the OS choice, the new computer as specified above comes from Concurrent Computer Corporation (CCUR) as a complete package named iHawk, bundled with the RedHawk OS. Achieving microsecond response time requires holding hardware components to a very high standard, and subsequently verifying the performance metrics in an appropriate environment. While the NSTX-U project certainly has the capability to specify, build, and test a real time platform in house, it is a cost and time saving measure to outsource that role to an organization that specializes in the industry. In this way, NSTX-U engineers can focus on the core business of plasma control.

3. New operating system

The NSTX Control System operated on a linux platform running the 32-bit version of Red Hat Enterprise Linux 4 (RHEL4). This operating system (OS) has reached end of life as of February 29, 2012. While this does not directly translate into an immediate technical need to change to a different version or vendor, it signifies a situation that both in theory and in practice is problematic. Newly discovered bugs remain unfixed, vendor software support becomes prohibitively expensive, and security vulnerabilities without updates add risk and violate policy. The hardware itself is unobtainable, the original vendor is out of business, and the drivers currently in use do not support a 64-bit platform or newer kernel. New hardware vendors no longer support the old OS, and the computer itself does not support new interface standards such as PCI Express. Upgrading the hardware to address these concerns and to add improved I/O performance requires a fully modernized system, as the aforementioned backwards compatibility is impractical. There is essentially a cascade of dependency resolution failures that forces an extensive upgrade should even the smallest attribute of the overall system change. This situation is far from ideal, especially in a research environment where the ability to rapidly deploy new ideas is paramount.

In evaluating OS possibilities, two main distributions proved suitable for consideration: Red Hat MRG and Concurrent RedHawk. Both of these distributions have the same underlying configuration principle: take a base OS, RHEL, and replace the kernel package with a custom version that incorporates real time patches and other associated changes. However, they do not use the same real time patches or baseline kernel version, and the accompanying support system is quite different.

The heart of the MRG product is the CONFIG_PREEMPT_RT patch [12–14]. This patch enables a new option for a real time application to preempt a kernel lock that might run for hundreds of milliseconds, which can be an eternity for fast real time applications. The obvious benefit to this approach is that it allows a user space real time application to push the kernel out of the way in favor of some deterministic event. Unfortunately, the increased determinism comes at the cost of overall system throughput. One such example of added latency due to this design is in the kernel spin locks that were replaced with mutexes. On our system as described in Section 2, it takes 67 μs to release a kernel mutex that is in contention. That cost can be unacceptably high, such that the real time application misses the deterministic event deadline anyway. In NSTX-U, that deadline is 200 μs, in which we have little room for delays. Our low level input card drivers need to sample data and process inputs with jitter less than 1 μs. This proved impossible on MRG, as jitter was anywhere from 20 μs to 50 μs. Problems like this combined with the overall complexity of the preemption design have prevented the patch set from gaining acceptance in the mainline Linux kernel since its creation in 2004, and ultimately in the NSTX-U project.

RedHawk takes a simpler approach. Instead of trying to interfere with critical kernel threads, RedHawk allows "shielding" a given CPU core from any kernel activity at all. Kernel threads still run, but they do not share cores with the real time applications. Contrary to the system-wide PREEMPT_RT solution utilized by MRG, processor shielding applies to a subset of cores and applications depending on the actual real time needs. Shielded cores can block access by any or all of interrupts, other applications, or even the system timer itself. Given that a typical real time solution involves distinguishing between "the real time application" and "the rest of the system", the fine grained control afforded by shielding naturally separates the two components. RedHawk takes this concept a step further and allows shielding of Non-Uniform Memory Access (NUMA) memory nodes as well, such that a given application stays within the memory directly attached to its assigned CPU.

Because the NSTX-U computer has a large core count, the RedHawk approach is easy to integrate. With 64 cores available to allocate between the kernel, real time applications, and support software, there is plenty of headroom to guarantee proper isolation.

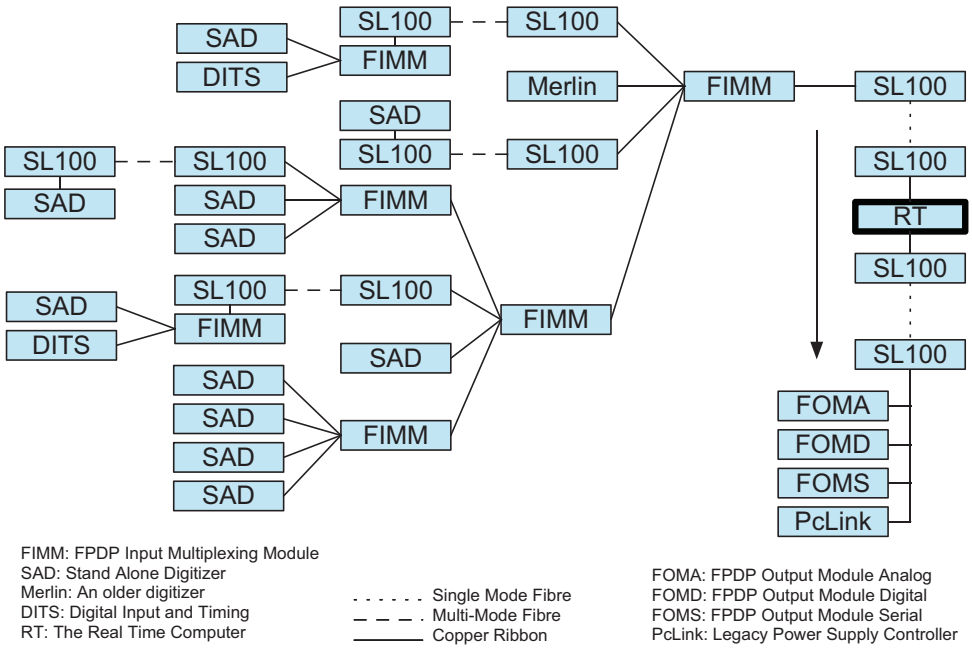

Fig. 1. Current NSTX Acquisition and Control System.

The current layout keeps the first 4 cores dedicated to the operating system, 20 cores allocated for plasma control, one core running nothing but a maximum priority emergency access shell that normally sits idle, and the rest for a new Digital Coil Protection System [24] running in tandem across its own set of 24 cores. It is trivial to change the layout to experiment with various configurations using the tools included with RedHawk (see Section 5.1).

RedHawk includes a tool called NightTune that provides a graphical way to configure all aspects of the real-time settings, including affinity, shielding, and NUMA of not just user applications, but interrupts as well. Using this tool, NSTX-U demonstrated over 100 different configurations across several hours, leading to the final chosen configuration scheme with very few hurdles. The graphical displays provide visual feedback of the inner workings of the system with as much detail as required, down to the nanosecond if necessary. One such example was our discovery of the vastly different "background noise" on the two PCI busses on the motherboard. NightTune aided the effective slot selection for each card, since not all cards fit on the "quiet" bus. Efforts on MRG were far less forgiving in these areas, requiring additional time to implement each new idea. We opted for the tool that allowed us to focus on what we wanted instead of how to get there. The final solution was ultimately simple, despite many attempts at complex configurations. With the operating system packed into the lower cores, the IO card interrupts and delicate software inner loops on the higher cores, the cards themselves segmented bus-wise, and instrumentation to guarantee the system stays deterministic, the NSTX-U control system operates with the hard real-time determinism required of our feedback control mechanism: every 200 μs cycle takes between 199 μs and 201 μs, each time, every time.

RedHawk offers additional benefits over MRG in the form of a hardware Realtime Clock and Interrupt Module (RCIM) and a corresponding software Frequency Based Scheduler (FBS). Combining these two elements yields a framework to schedule threads based on external interrupts, a very high resolution clock onboard the aforementioned RCIM, or a software triggered event. Under this arrangement, Concurrent guarantees a process dispatch latency (PDL) of 15 μs in the worst case. However, testing on the new hardware with appropriate shielding shows repeatable timings under 1 μs with an upper bound of 2 μs while under a load heavy enough to significantly exceed anything possible during operation.

4. FPDP I/O upgrade

The NSTX Control System (NCS) has always employed the Front Panel Data Port (FPDP) protocol [3a] for moving data between all of the integral components: the acquisition system, the real time computer, and the controlled power supplies. Originally designed for low latency high speed communication between VMEbus devices, FPDP has since been extended to support a variety of platforms including PCI and PMC. The fast board-to-board interface used in the VME community has now been adopted as a system interface in microcomputer-based real time systems such as those employed by NSTX.

The data acquisition system for NSTX-U is located about 1 km away from the real time computer doing the actual feedback control, and the commands have to return the same distance. Further, the various acquisition devices making up the FPDP input stream are distributed across a wide area, requiring on the order of 100 m cable length runs between devices. Serial FPDP excels at reliable communication of arbitrary data sizes over varied distances, which is precisely the scenario on NSTX-U.

NSTX made use of two different standards: parallel FPDP over copper (pFPDP) for short distances and easy signal splitting, and serial FPDP over fiber (sFPDP) on a Curtiss-Wright SL100 card for the longer distances between groups of digitizers and to and from the real time computer, stepping back to parallel/copper at the back of the computer itself. An NSTX-developed FPDP Input Multiplexing Module (FIMM) allows combining pFPDP streams into a single stream, albeit with an increasing latency at each level. The system is very flexible, supporting a wide variety of heterogeneous components: multiple kinds of digitizers, input multiplexers, digital inputs, and multiple transmission types see Fig. 1.

NSTX-U will improve this situation as shown in Fig. 2 by eliminating cascading multiplexers, now that advances allow for multiple serial fiber connections directly off of the PCI bus in the computer itself. In the new configuration, the digitizers will still output pFPDP into an SL100 for conversion to sFPDP. However, the

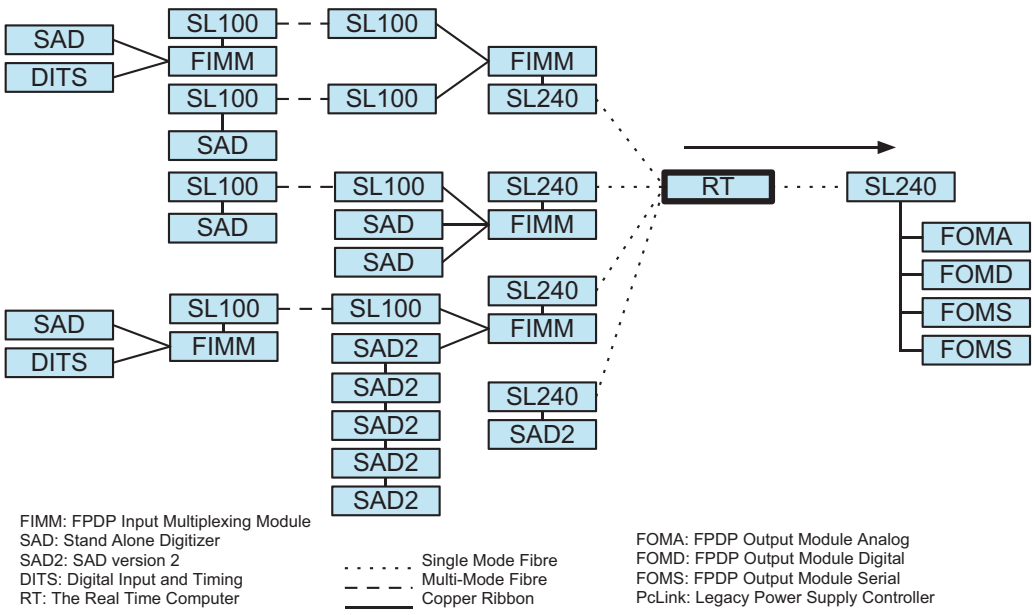

Fig. 2. Future NSTX-U Acquisition and Control System.

multiple levels of multiplexers will disappear, and multiple direct fiber runs will instead connect directly to the real time computer. This removes not only the multiplexing delays, but also the conversion back and forth between copper and fiber. Measured latencies amount to a total reduction from 69 μs to 25 μs. With a 5 kHz (or 200 μs) system periodic, this 44 μs savings represents a gain of more than 20%.

The switch to fiber is not without consequence, however. A definite benefit to a copper ribbon cable is the ability to easily split the signal out to multiple devices for use on a scope or other recording device. As mentioned in Section 2, previous iterations of the FPDP infrastructure utilized ribbon cables to mirror data between the real time control computer and a development system. This is still possible with fiber through the use of an optical signal splitter, but at a higher expense both in monetary cost and in signal loss. As a passive device, the splitter has a single input port and multiple output ports with various ratios of signal reduction totalling 100% of the original signal available. For reliability purposes, NSTX-U incorporates a Precision Rated Optics single mode splitter with a 50:50 ratio. However, ratios as imbalanced as 99/1 are available.

The FPDP output stream switched from copper pFPDP to fiber sFPDP as well, with a second SL240 mimicking the upgraded input side. New rectifier control hardware developed by NSTX and deployed as part of the upgrade removes aging components in use prior to the existence of NSTX, including replacing the old Merlin digitizers with the new Stand Alone Digitizer version 2 (SAD2). Previously, the FPDP output stream connected directly to CAMAC [21] "PClink" modules over fiber, which then broadcasted to rectifier control modules and ultimately to CICADA [22] control modules residing in the rectifier chassis. The new FPDP Output Module Serial (FOMS) [23] replaces this system entirely, receiving pFPDP data over existing copper lines and fanning out from a single transmitter to up to 8 channels of rectifier control over fiber.

5. Software improvements

5.1. Debugging

For NSTX-U, which often requires development on short timescales with hard deadlines, the need for efficient debugging is high. NSTX made use of a variety of locally developed tools and complex instrumentation procedures to gather data on problems and find and solve bugs. One tool examined and printed shared memory segments, another verified voltage readings of digitizers, and a third injected test data into the output stream. Instrumentation involved among other things, set ways to calculate timings down to CPU cycle counts for inner loops. These tools and procedures, while effective, swallowed up valuable resources and required a deep understanding of the system in its entirety to interpret results. NSTX-U will expand the old toolset with a commercial off the shelf (COTS) set of powerful debugging tools known as the NightStar Tools for Linux [15,16]. Alongside a traditional debugger, this toolset also includes a kernel tracer, a scheduling simulator, a real time performance tuner and more. For comparison, setting up a particular latency test on NSTX required instrumenting and recompiling code, delicate run conditions, and several days of lab time. The same test in the new environment required roughly 3 min and no code changes, while yielding a result more representative of actual runtime conditions.

To aid continual testing in parallel with normal operations, NSTX-U supports multiple kinds of MDSplus [20] trees for shot data storage. Shots created by the MDSplus data acquisition system during plasma operations, system testing, and calibration use the "engineering" and "eng_dev" trees to archive their data. As described in Section 2, there are redundant PCS platforms running simultaneously during the shot. The development computer with neutered output writes to the "eng_dev" tree to completely separate the data archives. During NCS software development with multiple developers, the need quickly arose to have a third tree without ties to the facility clock cycle. To avoid collisions during tree creation by multiple parties, NSTX-U employs a mechanism to request shots from the server atomically. A passkey protected daemon creates a new sequentially numbered shot in a separate tree called "eng_test" using a numbering scheme that is independent of the current NSTX-U shot number. The server daemon then sends the newly created shot id back to only the requestor via an MDS Event. Only one instance of the daemon can run and service requests at any time, thus preventing concurrency issues. On the requestor side, the waiting process has a timeout associated with it to account for error handling. The timeout is typically on the order of 6 s, as the server needs 5 s to fully create the tree for the shot.

5.2. Coding standards

Any moderately complex software package with multiple concurrent developers requires some level of standardization to ensure a consistent approach that is maintainable, extensible, and easily learnable by new developers. Central to this is a strict adherence to the rules present in one or more versions of the language standard. There are currently three versions of the C language, known colloquially as C89, C99, and C11.

C99, released in 1999 by the American National Standards Institute (ANSI) as ISO/IEC 9899:1999, was the first major update since the initial ANSI C language in 1989, known as both C89 (ANSI X3.159-1989) and C90 (ISO/IEC 9899:1990). Coupled with inline functions, refined scoping rules, and native IEEE 754 floating point hardware support, C99 is a viable starting point for real time determinism in a GNU/Linux computer. By inlining certain functions that existed as macros in NSTX, NSTX-U now benefits from compile time checking and additional optimizer opportunities without sacrificing speed. After rewriting code with scope in mind, the NSTX-U code base is more maintainable and more understandable. Finally, since the predominate data type for all I/O in the system is a 32-bit single precision float, the direct hardware interface to floating point operations provides an immediate, effective speedup.

Most important for fully utilized systems is the addition of the "restrict" keyword, something that allows compilers to optimize code equivalent to Fortran speeds. This is a potentially dangerous but substantially beneficial tool that serves as a contract between the user and the compiler that a given pointer has no aliases. The compiler will not verify the lack of aliasing. It will instead apply possible optimizations based on the unaliased claim of the user. If the developer misleads the compiler about the pointer restriction, the compiler is free to create optimizations that will break the program in subtly painful ways. Therefore, this is best reserved for those profiled bottlenecks with a well documented interface [11]. The significant places where NSTX-U received measurable speed improvements using this technique involved vector math functions and byte array manipulations.

C99 received a facelift in March of 2011 with the release of C11, a much more evolutionary rather than revolutionary change. C11 enables functionality needed for system stability such as concurrent threads and compatibility with current and future compilers utilizing these standards. New static assertions implemented in the NSTX-U debugging model help to catch bugs before they occur, and language-defined threading support makes the multi-threaded NSTX-U programs much easier to implement. Of most interest, however, is the built in atomic support that allows the compiler to enforce concurrent behavior. This allows proper mutex locking to take advantage of the atomic features in current processors. NSTX did not utilize locking in any form to control access to shared memory. While the system did work to some extent, it left much room for improvement, as such operations required non-deterministic system calls that introduced undesirable latency or produced erroneous results.

All other improvements to the system hinged on the ability to modernize the entire code base. Well written code is easy to modify, refactor, and forward port (for instance for the 64-bit upgrade). There were a number of simple programmatic rules that made this transition easier. First, we removed all code that made any assumptions about the sizes of data types. For example, it used to be customary in C programs to assume that sizeof(long) == sizeof(void *). C99 removed the need for such assumptions with the new data types intptr_t (an integer large enough to hold a pointer) and ptrdiff_t (an integer large enough to hold the difference of two pointers). Variables that required strict widths of 32 bits (such as for data acquisition) now utilize the new uint32_t, an unsigned integer type guaranteed to be exactly 32 bits wide. NSTX-U employs all of these current language standards, enforced at compile time.

5.3. 64-bit capable software

Updating old code to conform to the new C11 coding standards made forward porting to a 64-bit platform much easier. While it is true that a 64-bit GNU/Linux system can run 32-bit code, and it is also true that few plasma control programs need a 64-bit address space for accessing data, there are far more changes present in the x86_64 ABI that make a native 64-bit process desirable for real time plasma control. The number of general purpose registers doubled, and the increase allows function calls to pass more data via registers instead of on the stack. This enables optimizations that completely elide memory accesses altogether, especially in the case of small functions that for whatever reason are not amenable to inlining. The register count went up for the special purpose SSE registers as well, allowing for the natural acceleration of large matrix operations. The new platform is also safer, with advanced support for finer grained control over data and code memory regions. The ability to mark a memory region as non-executable provides better security from accidental buffer overruns that could cause entire shot failures; In the worst case, they could completely mask failures, instead yielding erroneous results.

5.4. Embracing the PCS code generator

General Atomics supplies a Plasma Control System (PCS) software package used in many fusion experiments [4–8]. NSTX-U will make use of their included code generator that makes writing an algorithm a substantially simpler task. It uses the compiler's C preprocessor to parse a list of configuration parameters and generate the requisite algorithm header include files that previously were entirely handwritten by NSTX. By incorporating this technology into NSTX-U, typical algorithm development time improved from an average of several weeks to several days. More importantly, it allows an algorithm developer to concentrate on the core functionality of the algorithm itself instead of the interface to the underlying framework.

In practice on NSTX-U, the source files maintained by hand shrunk in size by an average of 90%, with the removed code replaced by the files generated automatically. As a result, development time for new or modified algorithms dropped from weeks to days, testing and debugging of the algorithms themselves. Ultimately, the majority of development effort was spent on actual algorithm design, as more time became available to spend perfecting the details of how they worked. This is a marked change from past development cycles.

To use the code generator requires describing an algorithm using a customized language grafted on top of the C preprocessor. The pseudo-language is complex, but very well documented and thoroughly consistent within itself. For instance, because of the layers of nested languages, there are three different symbols used to separate arguments depending on the argument location. However, there is never any ambiguity as to which argument separator to use. The highest level arguments use commas, then vertical pipes, followed by semicolons at the lowest level of separation. This level separation is required when, for instance, a given argument is itself a list of arguments.

The basic construct used in an algorithm description is a GEN_ statement. There are several types, including GEN_ALGORITHM for the algorithm itself, and GEN_VECTOR for specifying some kind of input to an algorithm. Some GEN_ statements are shortcuts for a complicated GEN_VECTOR, such as GEN_MATRIX which will translate into several related GEN_VECTOR statements. The most common kind of input to an algorithm in the PCS system is the input

waveform, a function of time that provides some instantaneous value to a function. Subsets group related input waveforms together to provide the PCS operator with an organized interface. An input waveform has many parameters, most of which are optional. Some, such as the aforementioned subset, can inherit from a previously specified waveform to allow some amount of abstraction should a subset identifier change. Also useful is the ability to generate multiple GEN_VECTOR statements following some kind of pattern, such as replacing a token with either an incrementing number or the next item in a list of words.

6. Conclusions and future work

The increased capabilities of NSTX-U require numerous substantial changes to the Control System. While the physical changes to the computer are obvious, the less tangible changes needed to control it are immense. To that end we have improved upon a number of aspects of NSTX plasma control for NSTX-U. From a simplified FPDP hardware topology to a modernized approach to software development, the NCS in NSTX-U is a more maintainable and standardized system to effectively control an experimental device with a greatly increased operating envelope.

In the coming years, NSTX-U will explore using a GPU solution to the more computationally intensive aspects of plasma control. The biggest hurdles mainly involve the latency associated with getting data into and out of the graphics card along with the development learning curve for GPU programming. On the hardware I/O side, future plans may include bringing more FPDP connections directly into the real time computer, replacing SL100 cards in the field with faster SL240 versions, and upgrading all old digitizers to the new SAD2. Possible software upgrade solutions include using C++ in areas where it can reduce maintenance cost and increase runtime speeds.

References

[1] M. Ono, S.M. Kaye, Y.-K.M. Peng, G. Barnes, W. Blanchard, M.D. Cartera, et al., Nucl. Fusion 40 (2000) 557.
[2] J.E. Menard, S. Gerhardt, M. Bell, J. Bialek, A. Brooks, J. Canik, et al., Nucl. Fusion 52 (2012) 083015.
[3] (a) S. Gerhardt, Nucl. Fusion 52 (2012) 083012;
(b) D.A. Gates, J.R. Ferron, M. Bell, T. Gibney, R. Johnson, R.J. Marsala, et al., Nucl. Fusion 46 (2006) 17;
(c) E. Kolemen, D.A. Gates, C.W. Rowley, N.J. Kasdin, J. Kallman, S. Gerhardt, et al., Nucl. Fusion 50 (2010) 105010;
(d) E. Kolemen, D.A. Gates, S. Gerhardt, R. Kaita, H. Kugel, D. Mueller, et al., Nucl. Fusion 51 (2011) 113024.
[4] D.A. Gates, J.R. Ferron, M. Bell, T. Gibney, R. Johnson, R.J. Marsala, et al., Fusion Eng. Des. 81 (2006) 1911.
[5] D. Mastrovito, D. Gates, S. Gerhard, J. Lawson, C. Ludescher-Furth, R. Marsala, et al., Fusion Eng. Des. 85 (2010) 447.
[6] B.G. Penaflor, J.R. Ferron, M.L. Walker, General Atomic Co, A structured architecture for advanced plasma control experiments, Proc. 19th Symp. on Fusion Technology (Lisbon, Portugal) (16–20 September), 1996, p. 965.
[7] B.G. Penaflor, J.R. Ferron, R.D. Johnson, D.A. Piglowski, Fusion Eng. Des. 71 (2004) 47.
[8] S. Hahn, M. Walker, K. Kim, H. Ahn, B. Penaflor, D. Piglowski, et al., Fusion Eng. Des. 84 (2009) 867–874.
[9] S.P. Gerhardt, D. Mastrovito, M.G. Bell, M. Cropper, D.A. Gates, E. Kolemen, Fusion Sci. Technol. 61 (2012) 11.
[10] S.P. Gerhardt, R.E. Bell, A. Diallo, D. Gates, B.P. Le Blanc, J.E. Menard, Nucl. Fusion 53 (2013) 043020.
[11] M. Markus, Latin American Conference on Compiler Science, 2004.
[12] CONFIG PREEMPT RT Patch – RTwiki. 2010. 26 May 2013 https://rt.wiki.kernel.org/index.php/CONFIG_PREEMPT_RT_Patch
[13] A. Siro, C. Emde, N. Mc Guire, Proceedings of the 9th Real-Time Linux Workshop on November, 2007.
[14] B. Wolfgang, M. Cereia, I. Cibrario Bertolotti, Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on 22nd September, 2009, pp. 1–4.
[15] B. Jason, Real-Time linux: The RedHawk Approach. Concurrent Computer Corporation White Paper (Sep), 2008.
[16] Concurrent Real-Time Products – NightStar Tools, 27 May, 2013, http://real-time.ccur.com/home/products/nightstar-tools
[17] C. Pugh, T. Carrol, P. Henderson, 2011 IEEE/NPSS 24th Symposium on 26th June, 2011, pp. 1–5.
[18] Red Hat Network Reference Guide – Red Hat Customer Portal, 27 May, 2013, https://access.redhat.com/site/documentation/Red_Hat_Network/Reference_Guide/s1-sm-configuration.html
[20] J.A. Stillerman, T.W. Fredian, K.A. Klare, G. Manduchi, Rev. Sci. Instrum. 68 (1) (1997) 939–942.
[21] W.A. Rauch, Rev. Sci. Instrum. 57 (8) (1986) 1898–1900.
[22] N.R. Sauthoff, R.E. Daniels, Rev. Sci. Instrum. 56 (5) (1985) 963–965.
[23] J.E. Lawson, R. Marsala, S. Ramakrishnan, X. Zhao, P. Sichta, SOFE 2009, 23rd IEEE/NPSS Symposium on 1 June 2009, 2009, pp. 1–3.
[24] K.G. Erickson, G. Tchilinguirian, R. Hatcher, 2013 IEEE/NPSS 25th Symposium on 10 June 2013, 2013.