D2.3.1 – Operating systems at the extreme scale

WP2: Underpinning and cross-cutting technologies

Project Acronym	CRESTA
Project Title	Collaborative Research Into Exascale Systemware, Tools and Applications
Project Number	287703
Instrument	Collaborative project
Thematic Priority	ICT-2011.9.13 Exascale computing, software and simulation

Due date:	M18
Submission date:	31/03/2013
Project start date:	01/10/2011
Project duration:	36 months
Deliverable lead organization	UEDIN
Version:	1.0
Status	Final
Author(s):	Dan Holmes (UEDIN)
Reviewer(s)	Erwin Laure (KTH), Lorna Smith (UEDIN)

Dissemination level	
PU	*PU - Public*

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	27/02/2013	Draft version of the deliverable	Dan Holmes (UEDIN)
0.2	19/03/2013	Small revisions following review	Dan Holmes (UEDIN)
1.0	20/03/2013	Final version of the deliverable	Dan Holmes (UEDIN)

Table of Contents

Index of Figures

Index of Tables

1 Executive Summary

Standard commodity operating systems have evolved to serve the needs of desktop users and business application servers, which have very different requirements to HPC systems and applications. In general, commodity operating systems are not fit-for-purpose, even for current petascale machines, without extensive customisation.

The impact of operating system activities on application performance is not fully understood and is hard to predict. Many HPC systems are configured or customised by a trial-and-error approach, dealing with particular performance problems as they occur, rather than by applying a systematic method.

Specialised operating systems, developed specifically for HPC machines, trade rich functionality for high performance. Scalability is achieved by only implementing a subset of “normal” operating system services, which impairs the usability of the system by application programmers.

Design decisions for specialised HPC operating systems are often influenced by, and sometimes compromised by, design decisions for novel HPC hardware. One example is that the BlueGene/L hardware did not provide cache-coherency between the two processing cores in a node, which prevented the operating system from supporting shared-memory.

The desire to make specialised systems more usable encourages the re-introduction of functionality that can have a negative effect on performance and scalability. Thread scheduling was not supported by the BlueGene/P operating system but has been re-introduced in the BlueGene/Q operating system. This increases usability for application programmers but introduces a source of unpredictable load-imbalance that could reduce scalability, especially at extreme scale.

Specialised HPC operating systems have been continuously researched and developed for at least 20 years, driven (at least in part) by emergent trends in hardware design. Current systems demonstrate that excellent operating system scalability up to petascale is achievable. Although it is possible for major advances to be made in operating system development via disruptive technologies, currently there is no consensus on the direction required.

2 Introduction

The majority of systems in the most recent Top 500 list use the Linux operating system (OS); the most recent Top 500 list [1] includes 469 entries with OS family of Linux. As shown in Table 1, the average efficiency of the Linux systems is 66.6% (varying from 27.1% to 95.7%) whereas the average efficiency of all the non-Linux systems is 80.2% (varying from 58.9% to 93.4%).

OS Family	Average Efficiency (inclusive)	Average Efficiency (exclusive)	Count (inclusive)	Count (exclusive)
BSD Based	93.4%	67.4%	1	499
Linux	66.6%	80.2%	469	31
Mixed	83.1%	67.2%	7	493
Unix	79.3%	67.0%	20	480
Windows	75.5%	67.4%	3	497
All	67.5%		500	

Table 1: Efficiency of Machines in the November 2012 Top 500 List

Linux can be classified as a Full-Weight-Kernel (FWK) operating system. The variability in the efficiency of Linux systems reflects the wide range of customisations that are possible, from disabling some unneeded system services to modifications in the kernel code itself. It is reasonable to assume that customisations are only applied to an operating system if they improve performance (or improve usability without impairing performance) so the systems with very low efficiency (27%-50%) are likely to be standard distributions of Linux with minimal changes, or no changes at all.

All 7 of the systems with a "Mixed" OS family use Computer Node Kernel (CNK), which is a Light-Weight-Kernel (LWK) operating system that was developed by IBM specifically for Blue Gene machines. The "Unix" and "Windows" systems have similar average efficiency (79.3% and 75.5% respectively, see Table 1), which is higher than the average efficiency for Linux systems but lower than the average efficiency for CNK systems. The choice of a non-Linux OS for a Top 500 system is unusual and suggests that either the system designers or the system administrators have particular knowledge or expertise with the alternative OS, which means that the system is likely to be highly customised, possibly for a specific workload.

The operating systems on existing Top 500 machines are either standard (not heavily customised commodity products) – with low efficiency at scale – or non-standard (highly customised or designed specifically for that machine) – with much higher efficiency at scale. Various studies (e.g. [2], [3] and [4]) have shown that standard commodity operating systems do not allow scalable applications to scale well, even up to current system sizes, e.g. up to 10,000 nodes.

Section 3 of this document discusses and quantifies possible causes of poor scalability of applications that are due to the operating system.

Section 4 evaluates past and current developments in operating systems, highlighting scaling issues and attempts to address them.

Section 5 links HPC operating system issues to other areas within CRESTA.

Section 6 presents some concluding remarks.

2.1 Purpose

The purposes of this document are as follows:

- Quantify the potential impact of the operating system on applications at scale
- Evaluate and drive developments in operating systems to address scaling issues

2.2 Glossary of Acronyms

ABI	Application Binary Interface
ANL	Argonne National Laboratory
API	Application Programming Interface
BG	BlueGene
BGL	BlueGene/L
BGP	BlueGene/P
BGQ	BlueGene/Q
BSD	Berkeley Software Distribution
CIOD	Control and I/O Daemon
CL	Compute Library
CLE	Cray Linux Environment
CMT	Co-operative Multi-Threading
CNK	Compute Node Kernel
CNL	Compute Node Linux
CPU	Central Processing Unit
DMA	Direct Memory Access
FTQ	Fixed Time Quantum
FWK	Full-Weight Kernel
Gbps	Gigabit per second
GLIBC	GNU C Library
GPFS	General Parallel File System
GPGPU	General Purpose GPU
GPU	Graphics Processing Unit
HPC	High-Performance Computing
I/O	Input/Output
INK	I/O Node Kernel
IPC	Inter-Process Communication
LWK	Light-Weight Kernel
MPI	Message-Passing Interface
NFS	Network File System
OpenFOAM	Open source Field Operation And Manipulation
OpenMP	Open Multi-Processing
OS	Operating System
PEC	Power-Efficient Core
POP	Parallel Ocean Program
POSIX	Portable Operating System Interface [for Unix]
pthreads	POSIX Threads
QCDOC	Quantum Chromo-Dynamics On a Chip
QOS	QCDOC Operating System
SAGE	SAIC's Adaptive Grid Eulerian
SIP	Software Isolated Process
SLES	SUSE Linux Enterprise Server
SMP	Symmetric Multi-Processor
SPI	System Programming Interface
SSH	Secure Shell
STOC	Single-Thread-Optimized Core
TLB	Translation Look-aside Buffer
UDP/IP	User Datagram Protocol/Internet Protocol

3 The Causes of Poor OS Scaling

All operating system activities introduce some overhead to the execution time of applications. The noise or overhead generated by operating system activities can be amplified or absorbed by application activities in ways that are not fully understood and are difficult to predict.

3.1 System Calls

Some operating system activities are instigated by user applications and are necessary for their correct execution, e.g. system calls that control hardware resources such as a network card for communication between nodes. Here the overhead is a fixed cost per system call due to the context switch from user-code to kernel-code, which involves storing all CPU registers, trapping to the kernel, handling the system call, restoring the registers and waking-up the user process. Further overheads will occur when the system call causes the contents of the TLB or memory caches to be altered or 'polluted' with information that is not relevant to the application.

3.2 Interrupts

Other operating system activities are necessary to maintain system health and responsiveness, e.g. hardware interrupts will be generated by devices whenever an event occurs that cannot be handled without the help of a CPU. Here the overhead is a regular or irregular interruption for a fixed or variable length of time, depending on the type of event.

Each type of event produces an overhead with a particular noise signature, e.g. a timer interrupt may cause 2.5% overhead by responding to events at 1000Hz that take 25 microseconds for each response. The measured effects of various noise signatures can be much greater at scale than expected. Recent investigations [5] to characterise the sensitivity of representative applications to noise originating in the operating system kernel have measured a 30% slow-down for Parallel Ocean Program (POP) [6], caused by a 2.5% noise overhead (25 microseconds at 1000Hz), and a 50% slow-down for SAGE [7], caused by a 2.5% noise overhead (2500 microseconds at 10Hz).

This work also demonstrates that some codes are sensitive to the frequency of interruptions but relatively insensitive to the duration of those interruptions. It is suggested that this may be because the application activities 'resonate' with the OS kernel activities causing their effects to be 'amplified'. In addition, it is possible for applications to 'absorb' some of the noise, such as when some of the interrupts can be handled when the system would otherwise have been waiting for messages from other application nodes.

3.3 System Services

At present, the best way to deal with the effects of kernel noise in a FWK is to remove or minimise the sources of noise, i.e. by disabling system services that are infrequently used or unnecessary. Obvious examples from Linux are the print spooler daemon (because there is no printer attached to the compute nodes) and the mailer daemon (because the compute nodes will not send or receive email messages). It is also common practice to restrict or regulate the demands on key operating system services, such as virtual memory and scheduling, while an application is executing, by modifying the application code or by setting process parameters. Another approach is core specialisation where one or more cores are reserved for operating system services.

3.3.1 Virtual Memory

Frequent use of virtual memory by an application is undesirable because swapping pages of data between physical memory and permanent storage (e.g. a hard disk) is very slow. This can be avoided by the application not requesting more memory than is physically present and available in the node. However, the overhead of translating virtual memory addresses to physical memory addresses using the TLB cache cannot

be avoided by modifying application code (although some applications can be modified to make better use of TLB locality in a similar manner to optimising data-cache use).

3.3.2 Scheduling

Scheduling allows multiple processes to gain “fair” access to CPU resources. In desktop or server computers, there are typically many more processes than physical CPU cores. Complex algorithms are employed to determine which processes should be allocated time on which processors, depending on the process priority, its current state (e.g. waiting, busy or handling an interrupt), its recent activity, and many other factors. Frequently, a single busy process in a multi-core computer will be shifted between CPU cores by the scheduling algorithm to balance the load across all CPU cores. Generally, this causes poor use of memory caches and reduces the efficiency of the application. This can be avoided by setting an affinity mask for each application process (and thread, if used): a restriction that only allows the scheduling algorithm to consider certain CPU cores (e.g. a single core) for that particular process (or thread). However, the overhead of only allocating short time-slices and re-scheduling the process between each time-slice cannot be avoided by modifying the process parameters or the application code.

4 Past and Present HPC Operating System Developments

Currently, there are three approaches to producing an operating system for large-scale HPC machines:

1. Start with a standard operating system – a full-weight kernel (FWK) – and strip it down by removing unnecessary system activities and services.
2. Build a new minimal operating system – a light-weight kernel (LWK) – with limited functionality, usually for specific hardware.
3. Start with two existing operating systems – a FWK and a LWK – and merge them by modifying both so that they can inter-operate.

A fourth approach, based on a micro-kernel design combined with verifiably safe code, offers a potentially disruptive technology.

4.1 Single light-weight kernel

Development of light-weight kernels at Sandia National Laboratories began in 1991 with SUNMOS. Previous work in this field, e.g. Amoeba, Mach and Chorus [8], referred to the OS as a micro-kernel. In 1994, SUNMOS [9] was enhanced by Sandia Labs and renamed Puma [10], which included the first implementation of the Portals communication architecture. In 1996, Intel marketed Puma as a product called Cougar [9]. In 1997, the Portals communication was separated from Puma and became an independent component. In 2002, Sandia Labs created Catamount LWK [11] by porting Cougar to Red Storm (a prototype of Cray's XT series of machines). Catamount LWK has since led to Catamount N-Way (CNW) [12], which has support for multi-core CPUs and was licensed to Cray, and to OpenCatamount [13], which is a free open-source version released by Sandia Labs. A further LWK, called Kitten [14], is currently being developed at Sandia National Laboratories to aid research into how to better use multi-core processors and hardware virtualisation.

The LWKs from Sandia statically allocate main memory with a large page size such that the TLB entries are fixed addresses and the majority of them fit into the TLB cache. This reduces the overhead of a virtual address space by reducing the number of TLB cache misses, which would normally cause a 'page-fault', i.e. a significant but unpredictable delay in accessing the requested memory location. There is also no support for virtual memory, which eliminates the overhead of paging to and from disk. Another advantage of static allocation of main memory is that the operating system will not move any memory pages – each virtual address will always translate to the same physical address. This means hardware devices that require physical memory addresses (e.g. an Infiniband device requires the physical address for the input and output buffers) do not require the memory page to be 'registered' or 'pinned'.

Until the development of Catamount N-Way in 2008, the LWKs from Sandia did not include multi-core support; they supported only one single-threaded user-mode application process per node at a time. CNW introduced SMARTMAP [15], a mechanism for implementing shared-memory by using fixed virtual address offsets to directly access the memory of other processes. This is similar to how memory is shared between threads within a single process. In that case, the virtual address space for the process is inherited by all threads within that process so that all process memory is accessible by all process threads. With SMARTMAP, the physical memory location of the virtual address space of all processes is known by all processes. The entire system memory is accessible to all processes although there is a distinction in each process between the memory allocated to that process and memory that is allocated to other processes. The similarity to multi-threaded processes extends to both the advantage that direct memory access can be faster and the disadvantage that code must be 'thread-safe', i.e. it must avoid race-conditions and simultaneous conflicting memory accesses. Using SMARTMAP is optional so that codes, or parts of codes, that are not thread-safe can still execute without errors.

The work on SUNMOS, Puma and Catamount by Sandia Labs influenced IBM's design and creation of a light-weight kernel, Compute Node Kernel (CNK), for the BlueGene series of machines. The BlueGene/L (BG/L) was a successor to the QCDOC [16] machines, for which a custom operating system, called QOS, was created. There are many similarities between QOS [17] and current light-weight kernels, including CNK. The QCDOC machines and its QOS operating system were designed specifically to run a single physics application from the field of computational quantum field theory. The operating system was deliberately restricted to support for a single process with a POSIX-like interface with UDP/IP network connections and NFS file-system. This model was also adopted for CNK on the BG/L (general availability 2004) [18] where all processes are single-threaded supporting only a subset of the full POSIX interface. The compute nodes on BG/L operate either in co-processor mode or in virtual mode, running either one or two single-threaded processes, respectively. The two cores in each BG/L node are not cache-coherent: all communication between processes is via MPI, which is implemented using communication facilities provided by CNK for the specific customised hardware networks. Multi-threading requires manual scheduling by user code.

The design for the BlueGene/P machine (general availability 2007) [19] included a quad-core CPU with cache-coherency that can act as a 4-way symmetric multiprocessor (SMP) in addition to the virtual node mode and dual mode, which are similar to virtual and coprocessor modes from the BG/L. Support was added for multi-threaded processes, using a POSIX thread library (pthreads) or via OpenMP. Although the "fork()" system call is not supported the "clone()" system call can be used to create a limited number of new threads, depending on the operating mode of the compute node.

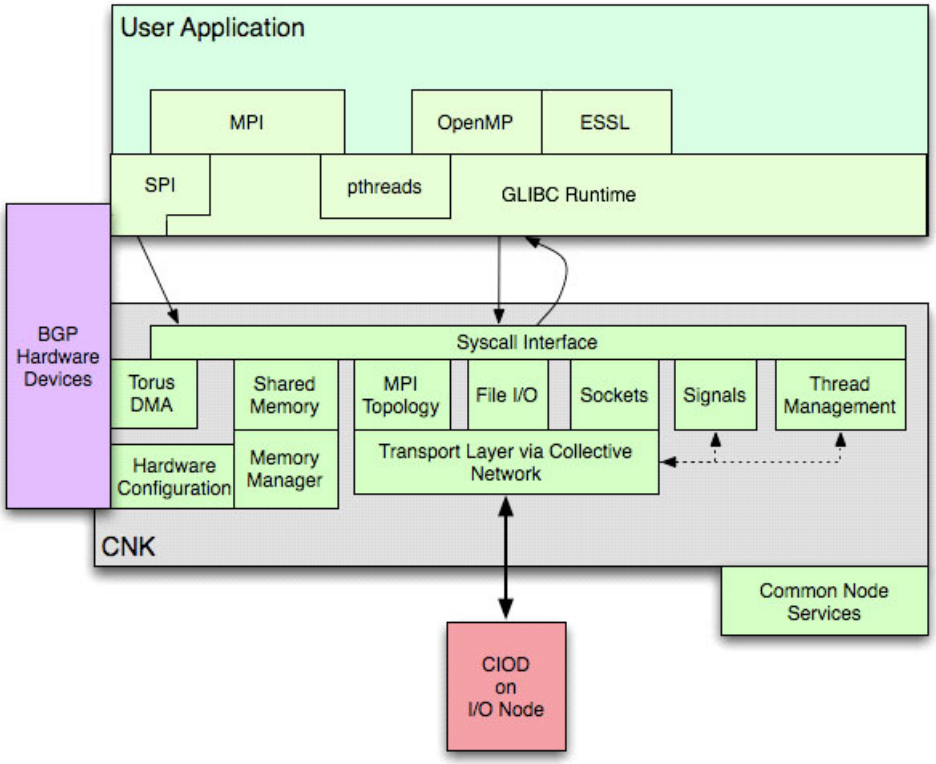

Figure 1: Overview of CNK Architecture in BG/P [19]

The CNK for BG/P (see Figure 1) directly supports the same system calls as the CNK for BG/L but additionally supports a large subset of standard Linux system calls, including file I/O, sockets and signals, by forwarding these requests from compute nodes to I/O nodes that run a full Linux operating system kernel (see Figure 2). Dynamic linking, whilst not supported for BG/L, is allowed for BG/P applications.

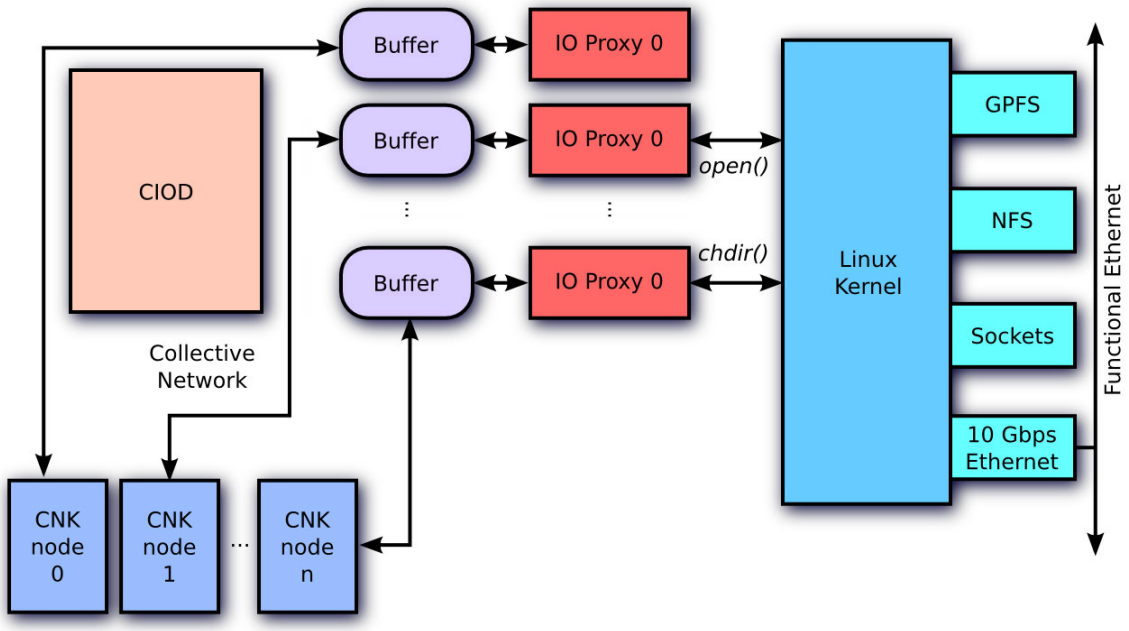

Figure 2: Overview of system call forwarding in BG/P [19]

Some potential customers saw the restricted functionality of CNK on BG/P as too limiting because it was difficult to re-engineer general applications that required the rich functionality of a full-weight kernel. Project Kittyhawk [20] at IBM Research explored the use of a micro-kernel, L4 [21], to support virtual machine instances capable of hosting general scalable applications. The application, the Linux operating system kernel and the required dependencies (system software and libraries) are captured in a system image. The image can be seen as a software appliance that is duplicated and deployed to as many compute nodes as needed to handle demand for the services provided by the application.

The most recent version of CNK was developed for the BG/Q machines (general availability 2012) [22]. The compute nodes in BG/Q contain 18 processor cores, each of which supports 4 hardware execution threads via Simultaneous Multi-Threading (SMT). Only 17 of these cores are active – the 18th is disabled and reserved to aid fault-tolerance (if one of the 17 cores fails, the 18th core is enabled and replaces the failed core). One of the active 17 cores is reserved for the CNK leaving 64 hardware threads on 16 active cores for user-level applications per compute node. Shared-memory and cache-coherency is available, enabling each node to act as a 64-way SMP. As with BG/P, system calls that cannot be directly handled by the CNK are forwarded to I/O nodes that run a full Linux operating system. However, more operating system functionality has been included in the CNK for BG/Q. Threading (hardware thread over-subscription) is now supported in CNK by a thread scheduler component. There is no time-slicing or time-quantum-based pre-emption; threads of the same priority are scheduled using a form of Cooperative Multi-Threading (CMT) with a “round-robin” ordering. Higher priority threads can cause pre-emption of lower priority threads and hardware interrupts may result in unbalanced dispatching of threads so that there is no guarantee that all threads of the same priority will make equal progress. This has the potential to re-introduce some of the problems normally associated with full-weight kernels including load-imbalance, kernel-induced noise and the amplification of these overheads for large-scale applications.

4.2 Single full-weight kernel

Four of the machines in the current Top 500 list use a Microsoft Windows operating system. Two entries list the OS as Windows HPC 2008 (ranks 132 and 183), one lists the OS as Windows Azure (rank 165) and one entry includes “Linux/Windows” in the

description of the machine – although the OS is listed as Linux (rank 17). Microsoft operating systems have a significant market share in some sectors and the HPC and Azure versions of Windows are therefore a natural choice for some supercomputer owners. However, being closed-source and proprietary software, manufacturers cannot enhance it to take advantage of novel hardware nor can system developers adapt it to better support a particular application or programming model. More customisation options are provided in the recently released Windows Server 2012 than were available in the previous version, Windows Server 2008.

Linux has displaced UNIX and, subsequently, all other operating systems to become the dominant choice for machines in the Top 500 list. It can be deployed almost entirely unchanged or it can be extensively customised. It has many advantages but, from a scaling point-of-view, it is challenging to achieve performance results commensurate with light-weight kernels. Of particular note is Cray Compute Node Linux (CNL), which is part of Cray Linux Environment (CLE) – the operating system for the top-ranked system in the current Top 500 list (as well as 17 other entries). This offers the programmer a familiar Linux environment, based on SUSE Linux Enterprise Server (SLES), with a stripped-down, low-noise Linux kernel on the compute nodes.

ZeptoOS [23] is a HPC operating system research project at the Argonne National Laboratory with a working implementation for BG/P. It is intended to be "the small Linux for big computers" and is based on an optimised compute node kernel derived from a standard Linux kernel combined with a kernel for I/O nodes derived from the I/O Node Kernel (INK) for BG/P. ZeptoOS enables more of the features of standard Linux, e.g. the SSH daemon is enabled on I/O nodes and a user can connect from an I/O node to compute nodes via telnet, requiring a telnet daemon in the compute node kernel. This increases the risk that background kernel activities will cause a noise signature that is amplified at scale by some applications. The telnet connection can be used to attach a debugger to one or more executing compute node processes. However, if the support and use of telnet in the compute node kernel causes amplified noise that slows the application execution by a factor of 20 or more (as in the POP application example in section 3.2), then the information gathered by the debugger may be rendered useless.

4.3 Multiple kernels

FusedOS [24] is a new HPC operating system currently under development by IBM with a prototype for BG/Q machines. It combines the two traditional approaches to HPC operating system development by fusing a LWK and a FWK (the architecture is shown on the left of Figure 3). IBM has created a user-level LWK, called Compute Library (CL), which is a port of CNK that is suitable for execution in user-mode rather than supervisor-mode. This is fused with a standard Linux kernel that is slightly modified to support the interaction of CL and Linux. FusedOS distinguishes two types of processor: a Single-Threaded-Optimised Core (STOC), which has the full capability of a modern CPU core, and a Power-Efficient Core (PEC), which may have restricted capability (such as a GPGPU core) and therefore may not be able to support a fully functional operating system. The execution model on PECs (shown on the right of Figure 3) is similar to normal compute nodes on BG/Q using CNK: the application code runs with full access to the hardware and no interference from the kernel or other application processes until it makes a system call. All system calls from an application running on a PEC are forwarded to a CL running as a user-mode process in Linux on a STOC, which handles the system call and returns execution back to the PEC. This closely resembles the model of a system call on a BG/Q compute node being forwarded by the CNK to the INK on an I/O node, which handles the system call and returns control back to the compute node.

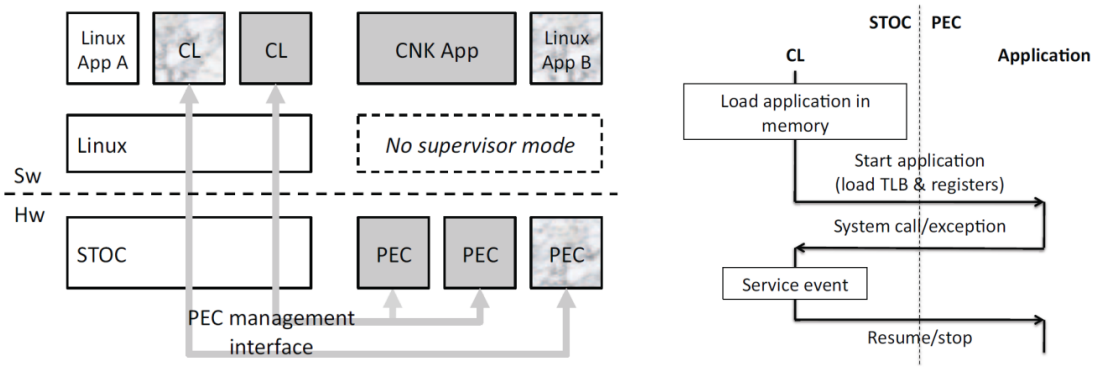

Figure 3: FusedOS architecture (left) and PEC management interface (right) [24]

The results (presented in Figure 4) from the Fixed Time Quantum (FTQ), or Selfish Benchmark [25], demonstrate that application processes running on PEC processors in FusedOS are given a higher proportion of compute cycles than processes running on the same hardware but in a customised HPC-ready Linux. As expected, code running on PEC processors experiences no interference from kernel activities, which eliminates the possibility of kernel noise being amplified at scale.

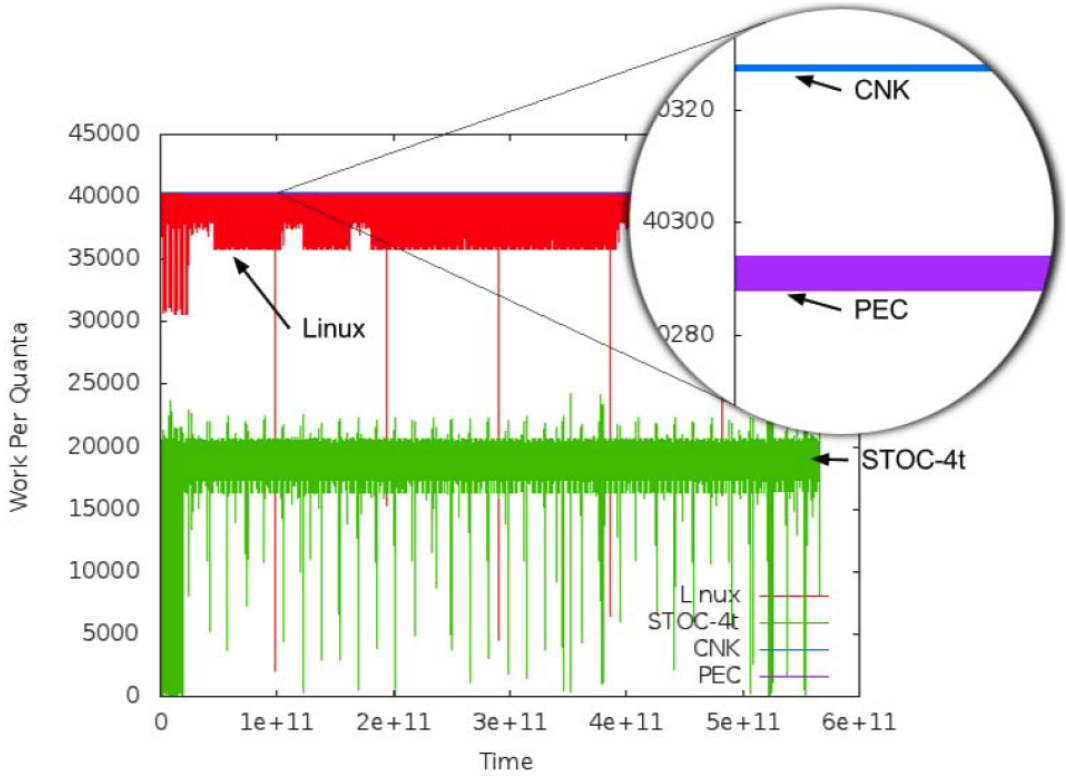

Figure 4: Performance of FTQ benchmark in different operating environments [24]

4.4 Disruptive Technologies

Microsoft are currently developing Singularity [26], a research operating system that may replace the NT kernel in future releases of the Windows product-line. The source-code for several different variations of this OS has been released under a shared-source license. Singularity is written pre-dominantly in safe managed-code (using multiple .Net languages [27][28]) and requires that all applications are also written in safe managed-code. This increases the reliability and verifiability of the kernel, drivers and all application binaries both at installation and at runtime. However, from the point of view of the HPC community, this requirement represents a major shift in both programming language and programming paradigm.

Using code that is verifiably safe at compile-time allows Singularity to rely on software-isolated processes (SIPs) rather than hardware protection for process isolation. This can reduce the cost of system calls and all other inter-process communication by up to 33% [29]).

Singularity is not currently intended to be an operating system for a distributed-memory HPC machine – although many of design goals overlap with current goals in exascale HPC research, in particular fault-tolerance and efficient support for multi-core, many-core and hybrid hardware architectures.

Singularity can be configured as a true micro-kernel (similar to MINIX 3 [30]), a monolithic kernel (similar to the current Windows NT kernel or the Linux kernel) or as a modular kernel (where some trusted services are allowed to execute in the same protection domain as the core kernel). The configurable nature of the kernel allows deployment as either a LWK or a FWK. The micro-kernel design means the multiple kernel SIPs could be run on dedicated, specialised processor cores (like the 17th core in a BG/Q machine). This provides a potential use for spare cores in “fat nodes” [31]. Alternatively, each node could dynamically load and unload parts of the OS, adapting to the needs of each phase of an application, e.g. by transforming from a LWK to a FWK and back again. Rather than treating GPGPU processors as functional units of a CPU, Singularity research is exploring their use as first-class OS-schedulable processing units.

It should be possible to extend the IPC mechanism in Singularity (contract-based message-passing channels) to support distributed-memory inter-node messaging and thereby provide an efficient route to implement MPI. An MPI library for Singularity would need to be written entirely in a .Net language using safe managed-code, such as the research project called McMPI [32].

5 Links to Other Areas within CRESTA

At least one of the CRESTA co-design applications, OpenFOAM [33], requires dynamic library linking but the "dlopen()" system call is not supported in many LWKs, i.e. Catamount and CNK (before BG/Q). The CNK for BG/Q compute nodes supports dynamic linking, with some restrictions, by enlisting the help of the full Linux kernel on the I/O nodes. This functionality may be supported in future LWKs, in particular FusedOS.

All of the CRESTA co-design applications (as well as most HPC codes that scale well on distributed-memory systems) use the message-passing programming model and therefore require efficient inter-process communication via a high-performance MPI library. Efficiently implementing MPI messaging for local messages, i.e. between processes within a shared memory node, requires an efficient synchronisation mechanism. However, OpenMP performance is reported to be poor on FusedOS [24] because thread synchronisation is performed by a system call, which is handled by delegation to a processor that is running a full-weight kernel. This performance issue may be fixed in future by handling the synchronisation system calls locally, i.e. without the costly delegation.

Asynchronous algorithms use non-blocking MPI communications and repeatedly call the MPI_Test function to discover when new messages arrive, rather than using the MPI_Wait function. This programming approach assumes that progress is made by MPI, as required by the MPI standard [34], despite there being no function call that is obviously expected to be time-consuming. Implementing MPI so that it makes progress in this scenario is possible even when the operating system supports only single-threaded processes with a form of CMT for system calls. However, this requires that, whenever possible, some progress is made during all MPI calls, which means that the precise timing of the work-load is less predictable. It is possible that this unpredictability adversely affects the performance of asynchronous algorithms, especially at scale – although this hypothesis is difficult to test.

Extending OS functionality, e.g. with a power-management API/ABI/SPI [35], must be done carefully to avoid it becoming a new source of kernel interference. Dynamically varying processor clock-speed during application execution for power-management reasons may introduce load-imbalance that adversely impacts application performance, especially at scale.

The IESP Roadmap calls identifies the need to develop a framework for an exascale OS. There is currently no consensus on which of the approaches described in section 4, if any, is the correct one. However, the present trend is to maintain backward compatibility for existing application codes by supporting well-established APIs whilst re-implementing the functionality in novel ways. Work towards the goals expressed in the IESP Roadmap includes the static memory maps in Catamount and BG CNK, which facilitate explicit management of the memory hierarchy, and the power-management work in CRESTA, (e.g. deliverable D2.6.3), which researches strategies and mechanisms for power/energy management in exascale systems.

6 Conclusions

The current trend in operating system research and development of re-implementing existing APIs is likely to continue. However, this approach is incremental and driven by developments in hardware as well as the necessity to improve the operating system to make full use of current technologies. Unfortunately, improvements that enhance scalability of the operating system often reduce usability.

This method of operating system development will provide scalability for the immediate future but it is likely to be limited by the original design decisions of modern HPC technology. Developments in hardware, operating systems, programming models and programming languages are all interdependent, which leads to cyclical improvements rather than novel approaches. The abstractions that have held true for hardware for several decades are no longer adequate to describe modern hardware. For example, procedural languages such as C and FORTRAN, assume single-threaded, sequential processing and memory isolation enforced by hardware protection. Operating systems now depend on this hardware protection mechanism to isolate the memory spaces for different processes, which requires an expensive context-switch when transferring control from one process to another. This cannot be avoided unless a disruptive technology breaks the dependency by introducing a novel way to protect process memory spaces.

Similarly, disruptive technologies may be needed to solve other scalability and performance issues, in operating systems and hardware, without sacrificing usability.

7 Acknowledgements

The author would like to thank Jeff Hammond (IBM), Brian Barrett (Sandia National Laboratories) and Kevin Pedretti (Sandia National Laboratories) for their invaluable contributions to this report.

8 References

[1] TOP500.Org, “Top500 List - November 2012,” November 2012. [Online]. Available: http://www.top500.org/list/2012/11/. [Accessed 28 February 2013].

[2] K. B. Ferreira, R. Brightwell, K. Pedretti and P. Bridges, "The Impact of System Design Parameters on Application Noise Sensitivity," 2010. [Online]. Available: http://www.cluster2010.org/presentations/session%206/ferreira.pdf. [Accessed 2013].

[3] D. Pradipta and M. Vijay, "jitSim: A Simulator for Predicting Scalability of Parallel Applications in Presence of OS Jitter," *Lecture Notes in Computer Science, Euro-Par 2010 - Parallel Processing,* pp. 117-130, 2010.

[4] A. Nataraj, A. Morris, A. D. Malony, M. Sottile and P. Beckman, "The ghost in the machine: observing the effects of kernel operation on parallel application performance," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, Reno, Nevada, 2007.

[5] K. B. Ferreira, P. Bridges and R. Brightwell, “Characterizing application sensitivity to OS interference using kernel-level noise injection,” in *In Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC ’08*, Piscataway, NJ, USA, 2008.

[6] D. J. Kerbyson and P. W. Jones, “A performance model of the parallel ocean program,” *International Journal of High Performance Computing Applications,* vol. 19, p. 261–276, 2005.

[7] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman and M. Gittings, “Predictive performance and scalability modeling of a large-scale application,” in *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, Denver, Colorado, 2001.

[8] A. S. Tanenbaum, "A comparison of three microkernels," *The Journal of Supercomputing,* vol. 9, no. 1-2, pp. 7-22, 1995.

[9] R. Riesen, R. Brightwell, P. G. Bridges, T. Hudson, A. B. Maccabe, P. M. Widener and K. Ferreira, “Designing and implementing lightweight kernels for capability computing,” *Concurrency and Computation: Practice and Experience,* vol. 21, pp. 793-817, 2009.

[10] R. Brightwell, “The Puma Lightweight Kernel,” 1 September 2006. [Online]. Available: http://www.sandia.gov/~rbbrigh/slides/invited/puma-acl-seminar-slides.pdf. [Accessed 28 February 2013].

[11] S. M. Kelly and R. Brightwell, “Software architecture of the light weight kernel, Catamount,” in *In Proceedings of the 2005 Cray User Group Annual Technical Conference*, 2005.

[12] R. Brightwell, K. Ferreira, J. Laros, S. Kelly, C. Vaughan, K. Pedretti, R. Ballance, J. Tomkins, T. Hudson and J. VanDyke, “Catamount N-Way Lightweight Kernel,” 2009. [Online]. Available: http://www.sandia.gov/research/research_development_100_awards/_assets/documents/2009_winners/CNWFinal_SAND2009-1490P.pdf. [Accessed 28 February 2013].

[13] R. Brightwell, “Sandia OpenCatamount Home Page,” Sandia National Laboratories, 2005. [Online]. Available: http://www.cs.sandia.gov/~rbbrigh/OpenCatamount/. [Accessed 28 February 2013].

[14] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Z. Cui, L. Xia, P. Bridges, A. Gocke, S. Jaconette, M. Levenhagen and R. Brightwell, “Palacios and Kitten: New high performance operating systems for scalable virtualized and native supercomputing,” in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, 2010.

[15] R. Brightwell, K. Pedretti and T. Hudson, “SMARTMAP: Operating system support for efficient data sharing among processes on a multi-core processor,” in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, 2008.

[16] P. Boyle, D. Chen, N. Christ, M. Clark, S. Cohen, C. Cristian, Z. Dong, A. Gara, B. Joó, C. Jung, C. Kim, L. Levkova, X. Liao, G. Liu, S. Li, H. Lin, R. Mawhinney, S. Ohta, K. Petrov, T. Wettig and A. Yamaguchi, "The QCDOC Project," *Nuclear Physics B - Proceedings Supplements,* vol. 140, pp. 169-175, 2005.

[17] P. Boyle, D. Chen, N. Christ, M. Clark, S. Cohen, C. Cristian, Z. Dong, A. Gara, B. Joó, C. Jung, C. Kim, L. Levkova, X. Liao, G. Liu, R. Mawhinney, S. Ohta, K. Petrov, T. Wettig and A. Yamaguchi, "Hardware and software status of QCDOC," *Nuclear Physics B - Proceedings Supplements,* Vols. 129-130, pp. 838-843, 2004.

[18] J. Moreira, M. Brutman, J. Castaños, T. Engelsiepen, M. Giampapa, T. Gooding, R. Haskin, T. Inglett, D. Lieber, P. McCarthy, M. Mundy, J. Parker and B. Wallenfelt, "Designing a highly-scalable operating system: the Blue Gene/L story," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, Tampa, Florida, 2006.

[19] IBM Corporation, "IBM Redbooks | IBM System Blue Gene Solution: Blue Gene/P Application Development," 2009. [Online]. Available: http://www.redbooks.ibm.com/abstracts/sg247287.html. [Accessed 28 February 2013].

[20] J. Appavoo, V. Uhlig and A. Waterland, "Project Kittyhawk: building a global-scale computer: Blue Gene/P as a generic computing platform," *ACM SIGOPS Operating Systems Review,* vol. 42, no. 1, pp. 77-84, 2008.

[21] J. Liedtke, "Toward real microkernels," *Communications of the ACM,* vol. 39, pp. 70-77, 1996.

[22] IBM Corporation, "IBM Redbooks | IBM System Blue Gene Solution: Blue Gene/Q Application Development - Update," 2013. [Online]. Available: http://www.redbooks.ibm.com/redpieces/abstracts/sg247948.html. [Accessed 28 February 2013].

[23] Argonne National Laboratory and the University of Oregon, "ZeptoOS: The Small Linux for Big Computers," Argonne National Laboratory, [Online]. Available: http://www.mcs.anl.gov/research/projects/zeptoos/. [Accessed 28 February 2013].

[24] Y. Park, E. Van Hensbergen, M. Hillenbrand, T. Inglett, B. Rosenburg, K. Ryu and R. Wisniewski, "FusedOS: Fusing LWK Performance with FWK Functionality in a Heterogeneous Environment," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*, 2012.

[25] P. Beckman, S. Coghlan, B. Gropp, R. Lusk, K. Yoshii, A. Malony, S. Shende and S. Suthikulpanit, "ZeptoOS: Small Linux for Big Computers," 2005. [Online]. Available: http://www.cs.unm.edu/~fastos/05meeting/ZeptoOS-Beckman.pdf. [Accessed 28 February 2013].

[26] Microsoft Research, "Singularity," 2013. [Online]. Available: http://research.microsoft.com/en-us/projects/singularity/. [Accessed 28 February 2013].

[27] ISO, "ISO/IEC 23270:2006 Programming Languages C#," 26 January 2012. [Online]. Available: http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=42926. [Accessed 28 February 2013].

[28] M. Fähndrich, M. Aiken, C. Hawblizel, O. Hodson, G. Hunt, J. R. Larus and S. Levi, "Language Support for Fast and Reliable Message-based Communication in Singularity OS," *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006,* vol. 40, no. 4, pp. 177-190, October 2006.

[29] G. C. Hunt and J. R. Larus, "Singularity: rethinking the software stack," *ACM SIGOPS Operating Systems Review - Systems work at Microsoft Research,* vol. 41, no. 2, pp. 37-49, April 2007.

[30] A. S. Tanenbaum and A. S. Woodhull, Operating Systems Design and Implementation, 3rd Edition ed., Pearson, 2006.

[31] CRESTA, “D2.4.1 Alternative use of fat nodes,” 2013.
[32] D. J. Holmes, McMPI: A managed code message passing interface, Edinburgh: University of Edinburgh, 2012.
[33] OpenCFD Ltd, “OpenFOAM - The Open Source Computational Fluid Dynamics (CFD) Toolbox,” 2013. [Online]. Available: http://www.openfoam.com/. [Accessed 28 February 2013].
[34] MPI Forum, “MPI Standard,” 2012 September 2012. [Online]. Available: http://www.mpi-forum.org/docs/docs.html. [Accessed 28 February 2013].
[35] CRESTA, “D2.6.3 Power measurement across algorithms,” CRESTA, 2013.