

# Full Operating System Core Specialization

---

Larry Kaplan, Cray Inc. ([lkaplan@cray.com](mailto:lkaplan@cray.com))

Operating system (OS) interference with high performance application computation and communication is a significant concern for HPC system designers, especially at large scale (Beckman, Iskra, Yoshii, & Coghlan, 2006). The common use today of Bulk Synchronous Programming styles where all processing elements of an application must periodically synchronize in some form (e.g. via MPI collectives) can increase the detrimental effect of OS interference, sometimes called “noise” (Beckman, Iskra, Yoshii, & Coghlan, 2006a). At odds with the desire for minimal OS interference is the need applications often have for diverse OS services.

This paper proposes research to address the OS interference issue while at the same time maintaining the services that many applications require from the OS. This is an important balance to maintain to provide for wide applicability of HPC systems and their OS implementation. While there is some concern as to the number of different applications that will be able to scale to exaflops performance levels, the fewer impediments in the way of that scaling, the more applications that will have a chance to scale. Similarly, the fewer restrictions on the services provided by the system, the more applications that system will be able to support.

## Description of Research Direction

A common solution for addressing OS interference is the use of a lightweight kernel (LWK)(Lange, et al., 2010) (Giampapa, Gooding, Inglett, & Wisniewski, 2010). However this solution often does not provide the OS functionality required for supporting a wide range of applications and their needs for various services. Work has been done with virtualization to provide both a regular OS kernel such as Linux along side an LWK on the same node. However one or the other OS runs in a virtual machine. Even with para-virtualization, this can add unnecessary overheads, especially to the communication paths.

The Cray Linux Environment (CLE) currently provides a feature called Core Specialization, which allows the user to ask that the cores on the nodes for their job be split into two groups, one for running the application and one for running system services (Cray Inc., 2012). Even with only a handful of cores per node (12-16), this feature has shown performance benefits for some applications (Oral, et al., 2010). We expect the situation to only improve as more cores are available within each node and a smaller fraction of the cores is therefore needed to provision the necessary services.

Today these cores still run an identical kernel as a single OS instance. This paper proposes research to bring together the above concepts to allow both a heavyweight kernel such as Linux and a lightweight kernel to run simultaneously on different cores of the same node, without relying on techniques such as virtualization, which can add overhead. Previous research has mostly used completely different kernel bases for this work, as opposed to Cray’s current implementation that uses identical kernels. The research would explore this spectrum, and also determine if some middle ground is possible, using kernels derived from the same base, but configured and built differently. The appropriate number of OS instances would also be considered.

We also note that on today’s GPU enabled systems, no real OS runs on the GPU. Instead, a host processor based kernel driver controls it, which can often be a bottleneck for fine-grained scheduling. Having some form of LWK or trusted runtime on the GPU could also be enabled by this research. This will be relevant to future HPC systems that combine both traditional and GPU-style processing elements within the same ASIC or die.

Some of the areas of focus for this research would include techniques for integrating and coordinating the disparate OS activities, fine-grained scheduling across the images, provisioning of services from the heavyweight kernel to the

LWK, determination of the proper location for provisioning individual OS services, and software engineering issues for maintenance and build of a heterogeneous OS solution, to name a few.

## Challenges Addressed

**OS interference:** By segregating application computational activities onto processing components that are distinct from those providing services, the application will experience less (ideally, no) interference from service activities.

**Scalability:** By reducing or eliminating local OS interference on each processing element, aggregate operations such as barriers and collectives will perform more consistently and with lower total latency (due to not needing to wait for processing elements that were interfered with by the OS).

**Service Provisioning:** Applications need services, some applications more than others. This work allows for full service functionality to be provided without all of the interference normally associated with such services.

## Maturity

This is not a completely novel research direction. Successes have been demonstrated with Cray's current core specialization where some applications with frequent collectives run better with fewer computational cores and a dedicated "service" core. But there are many more candidate applications for which this should be true and currently is not. Virtualization based research into this technique has also shown some advantages, but with overheads that could be reduced, especially in the ability to do fine-grained scheduling between the OS instances and in access to the communication hardware. There is also significant difficulty in today's GPU-based systems for managing those devices and the tight coordination of those processing elements with more traditional ones.

## Uniqueness

The desire for scalable computations to proceed unimpeded by service provisioning appears to be unique to HPC and exascale. In the data server world, most applications appear to rely heavily on OS services and/or are not so tightly coupled or synchronized that OS interference has a significant effect. Tightly coupled and scalable applications really are the hallmark of HPC. To the extent that "mainstream" applications move in this direction, the concerns may be shared.

## Novelty

The main novelty of this research is the resulting tight coupling between disparate operating systems running on the same ASIC or die. Existing solutions today either have too much overhead, or not enough differentiation between kernels to achieve the goals of scalable computation along with effective standard service provisioning.

## Applicability

If successful, this type of hybrid or heterogeneous operating system could be used effectively on other systems, especially if their application space starts experiencing some of the addressed concerns.

## Effort

The exploration of this functionality has two major areas of focus plus several related areas. Good progress could be made with approximately 4 FTEs for several years.

## Works Cited

Beckman, P., Iskra, K., Yoshii, K., & Coghlan, S. (2006). Operating System Issues for Petascale Systems. *SIGOPS Operating Systems Review*, 40(2):29–33.

Beckman, P., Iskra, K., Yoshii, K., & Coghlan, S. (2006a). The Influence of Operating Systems on the Performance of Collective Operations at Extreme Scale. *IEEE International Conference on Cluster Computing* (pp. 1-12). IEEE.

Cray Inc. (2012). *Workload Management and Application Placement for the Cray Linux Environment*. Cray Inc.

Giampapa, M., Gooding, T., Inglett, T., & Wisniewski, R. (2010). Experiences with a Lightweight Supercomputer Kernel: Lessons Learned from Blue Gene's CNK. *SC 2010*. IEEE.

Lange, J., Pedretti, K., Hudson, T., Dinda, P., Cui, Z., Xia, L., et al. (2010). Palacios and Kitten: New high performance operating systems for scalable virtualized and native supercomputing. *IPDPS 2010*. IEEE.

Oral, S., Wang, F., Dillow, D., Miller, R., Maxwell, D., Henseler, D., et al. (2010). Reducing application runtime variability on Jaguar XT5. *CUG 2010*. CUG.