Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Projektarbeit

Evaluation of Single System Images

vorgelegt von

Niclas Schroeter

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen

Studiengang:	Informatik
Matrikelnummer:	6926553
Betreuer:	Jannek Squar

Hamburg, 2022-08-17

Abstract

Applications in HPC often rely on OpenMP for parallelization. Due to OpenMP's reliance on shared memory and threads, its scalability is inherently limited. Multiple approaches to solve this scalability issue are part of today's research, among them are the Single System Image (SSI) kernels and operating systems. The contribution of SSIs to allow OpenMP applications to scale beyond a single compute node is the provision of distributed shared memory and thread migration across multiple nodes, though these features are not guaranteed to be properties of every Single System Image kernel.
This work evaluates the fitness of current SSI kernels for the problem, basing the results on the recent past of SSI kernels and Popcorn Linux, an experimental SSI kernel currently in development. Results indicate that these operating systems are not suitable to enhance the scalability of OpenMP applications without major drawbacks, instead the focus should be directed to other approaches, for example tool-based ones.

Contents

1 Introduction

In today's HPC landscape, many applications rely on OpenMP [9] for parallelization. This collection of compiler directives, environment variables and select library routines allows the developer to introduce multithreading on shared memory at a low entry barrier regarding the difficulty of implementation. Ever since version 4.0, offloading to accelerators, such as GPUs, is also supported through new directives. Due to this ease of use, the option to transform existing code incrementally and the vast support from most available compilers, many users without a background in computer science are able to efficiently parallelize their code with OpenMP. However, since OpenMP only allows for parallelization at a shared memory level, alongside the available accelerators in a given node, its scalability is inherently limited. Code that relies solely on OpenMP for parallelization can only be executed on a single compute node of any given HPC cluster. In order to exploit the massive amounts of compute power offered by such HPC systems, the mentioned OpenMP-reliant applications need to be adjusted, either explicitly or implicitly. An example for an explicit adjustment could entail the employment of a different programming model that supports distributed memory parallelization. The defacto standard for such a model is the Message Passing Interface (MPI) standard [13]. With MPI, multiple processes can compute in parallel while exchanging information in the form of messages. However, in order to incorporate MPI into an existing application, said application has to undergo a major transformation, without any options for incremental parallelization either. So while it is a possibility to transform existing applications from using only OpenMP to employing either just MPI or even a hybrid of MPI and OpenMP, such a transformation requires certain knowledge and skills that scientific developers without a background in computer science do not necessarily have, while also being very time-consuming, depending on the size of the application. The same is true for any other parallel programming model that requires fundamental transformations of existing code.

To aid with the transformation of OpenMP code to MPI code, several approaches investigated the automatization of this process. Tools were developed and evaluated, which transformed OpenMP applications at different stages, ranging from compile-time to runtime, showing promising results regarding the performance of the transformed applications. One example for such a tool is CATO (**C**ompiler **A**ssisted Source **T**ransformation of **O**penMP kernels) [24], which aims to transform the OpenMP code with the help of LLVM.

Another approach to solve the problem of scalability would be to abstract the distribution of memory into virtual shared memory, which would circumvent the need to fundamentally change the applications built with OpenMP. This concept is known as distributed shared memory. One of the options for introducing distributed shared memory is the employment of so-called Single System Images (SSIs). These systems create a single unified view of the underlying distributed computing resources, providing transparency in regard to resource management [10]. This abstraction can be introduced at different levels, ranging from software, over the kernel-level, down to the hardware.

This work focuses on the evaluation of Single System Image kernels, examining an exemplary open-source SSI operating system called Popcorn Linux [6]. The evaluation is focused on the usability of Popcorn Linux in regards to the problem introduced above, while also generalizing the findings towards an evaluation of the fitness of SSI kernels as a solution for this specific problem.

This work is structured as follows: Chapter 2 provides a brief overview on related work regarding distributed shared memory and Single System Images in general. Chapter 3 explains the concept of SSIs in more detail, as well as introducing Popcorn Linux and its properties. Finally, Chapter 4 contains the aforementioned evaluation, followed by a brief summary of the findings.

2 Related Work

The aforementioned tools for automatic transformation are currently topic of multiple researches. These approaches range from newly developed compiler techniques [8] to automated compiler-runtime systems [17]. Other approaches incorporate LLVM [18] to either analyze the bytecode of an application [14] and transform the code accordingly, or, as it is the case with CATO, transform the code during the optimization phase.

The concept of distributed shared memory (DSM) has been a topic in research since the 1980's. DSM has been implemented in both hardware and software. For hardware approaches, the goal was to create scalable multiprocessor architectures with shared memory and cache coherence. Two systems that achieved this goal were the Stanford DASH [19] and the Stanford Paradigm [11]. While HPC systems nowadays do not offer a single address space over all nodes, hardware-based DSM approaches are still relevant today in the form of the NUMA architecture, which will be further discussed in Section 3.1, alongside some software-based DSM approaches.

Many Single System Image operating systems can synonymously be referred to as DSM operating systems, although this generalization does not cover all of them, as DSM is not strictly a part of every definition of SSIs. The first Single System Images were created in the 1980's, their history and impact alongside their successors is analyzed in detail in Section 3.2. Within this category of SSI kernels, multiple comparisons regarding their performance were conducted, ranging from the usage as a web-grid built from desktop PCs [12] to feature comparisons when used in a cluster environment [20].

More recently, an overview of different SSI approaches, especially the SSI kernels was by provided by Healy et al [15], covering many of the released SSI kernels and their differences, focused mainly on the reasons for their failure on the operating system market.

Popcorn Linux itself has been evaluated in multiple papers, however only by the original developers. Many of them have been focused on the performance of Popcorn Linux when it is deployed in an environment with heterogeneous-ISA processors [5] [7]. Barbalace et al [6] also compared Popcorn Linux to a traditional SMP Linux with three kernels from the NAS Parallel Benchmarks (NPB), focusing their evaluation on comparing the performance of the OpenMP version on both operating systems and, independent of that, also comparing the performance of the MPI version of these kernels on the two operating systems. However, their evaluation was performed only on a single node, not covering the homogeneous-ISA setting across multiple nodes that this work is aimed at.

3 Single System Images

3.1 General

Buyya et al define a Single System Image as ”the property of a system that hides the heterogeneous and distributed nature of the available resources and presents them to users and applications as a single unified computing resource” [10]. In order for an entire system to appear as a complete SSI, multiple levels have to cooperate to supply the necessary properties. Multiple authors provide different categorizations of these levels, though all of them include a hardware level, a kernel or operating system level, and a user or application level to some extent. A perfect SSI would need cooperation from every level to provide all users with the services necessary to create the illusion of a single unified system, even though it is distributed. These services include, among others, a

Level	Example
Hardware	(cc)NUMA
OS / Kernel	Popcorn Linux, Kerrighed etc.
User	cJVM, SHMEM or other PGAS applications

Table 3.1: Overview of the different levels at which SSIs can be implemented. A perfect SSI would require each level to provide an implementation and cooperation between the layers.

single entry point. The user can log in to the cluster under one address and the system can transparently distribute said user to combat load imbalances. Another important service is a single point of control, where the user has control over each node in the cluster from a single shell or GUI. A perfect SSI would create an abstraction of the distributed memory of each node, presenting it instead as one single shared memory space. However, as of today no complete SSI has been achieved due to the complexity of coordinating all of the components mentioned above, while still maintaining a scalable system [23].

There exist different components in their respective levels that implement SSI features within their boundaries. An example for a hardware-based SSI is the NUMA architecture. Each processor is assigned a part of the local memory, but the entirety of said memory is still treated as shared memory by all levels above the hardware. These types of systems are also referred to as providers of distributed shared memory.

DSM can also be provided via software support, although the performance is worse compared to direct hardware support. Programming languages and libraries that support the Partitioned Global Address Space model (PGAS) [2] present the distributed memory of multiple nodes as one global shared memory space. Examples for PGAS languages include Unified Parallel C and Coarray Fortran, example libraries include the different SHMEM implementations. However, since their use in the problem described in this work would constitute a complete transformation of the underlying OpenMP code, these SSI-level programming languages and libraries are not considered further on their own, although they can be utilized in tools for automatic transformation of said OpenMP applications. Examples for software-based DSM in the form of runtime systems are multiple attempts at creating Java Virtual Machine implementations that provide an SSI of a cluster environment to a Java application, for example the cJVM [3] from 1999.

3.2 SSI Operating Systems and Kernels

This work focuses on the SSI operating systems and kernels. In a perfect Single System Image, this level is responsible for providing transparency regarding process migration to balance the load, managing access to certain files depending on whether a node-local file or a global file is needed, and many more.

The first SSI operating systems emerged in the 1980's, before the term was coined. Locus [25] and Plan 9 [22] built the foundation for the SSI operating systems to come. Locus was built as a distributed operating system with UNIX compatibility, enabling a view of multiple machines connected over the network as a single computer. This operating system later culminated in the Single System Image called OpenSSI. Plan 9 was built with the intention to move away from UNIX around the time that personal computers started to replace the previous monolithic timesharing systems. With this OS, users were able to create their own computing environment and use it on any machine in the network, utilizing concepts such as a single filesystem namespace or single process identification on distributed computing resources.

Until the 2000's, many different SSI operating systems emerged, such as MOSIX [4], Kerrighed [21] and Solaris MC [16]. Today however, work on most of these operating systems has been terminated. Kerrighed's last update was around 2012 and MOSIX became proprietary and received its most recent update in 2017, to name a few examples. Only very few are still maintained or actively being developed, among them are OpenVMS, a proprietary SSI which is focused on high-availability servers, and Popcorn Linux.

This development in the lifecycle of the SSI operating systems has been predicted by Buyya et al, citing development and maintenance costs that were too high compared to the market share of these operating systems, alongside the need to specifically develop all components of a cluster towards supporting SSIs, losing flexibility in the process. Due to the low availability of SSIs these days, this work focuses only on Popcorn Linux for all further evaluations.

3.3 Popcorn Linux

Popcorn Linux is a replicated-kernel operating system based on the Linux kernel. It is a project of the Systems Software Research Group at Virginia Tech. The goal of the project is to explore a different approach to the design of current operating systems, addressing potential operating system scalability issues. As opposed to a single kernel instance that manages all CPUs and other hardware resource, the replicated-kernel approach aims to have multiple kernel instances instead, either one per CPU or one per group of CPUs. These instances control different non-intersecting subsets of hardware and communicate via message passing over a newly introduced inter-kernel communication layer. This communication enables the coordination of multiple kernels, so that the abstraction of a single operating system, and therefore the abstraction of an SSI, can be maintained.

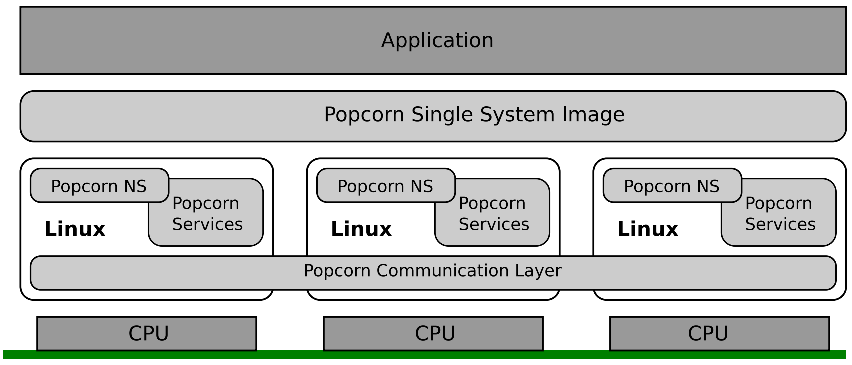

Figure 3.1: Popcorn Linux architecture on multicore hardware [6]. Each CPU is equipped with its own kernel, connected by the communication layer.

The inter-kernel communication layer is built on top of shared memory, utilizing private receive-only buffers, over which message passing occurs. The kernels themselves are fully operational with the resources assigned to them, the OS state is partially replicated on each kernel. Operating system services in Popcorn Linux are rewritten as distributed services to accommodate for the replicated kernels in the system. In order to balance the load on the system, Popcorn Linux enables migration of user space tasks, allowing for a transparent execution of applications across multiple kernels.

The Popcorn Linux architecture is schematically displayed in Figure 3.1, although it is not necessary to introduce one kernel per CPU. This operating system also allows for multiple CPUs to be managed by a single kernel, to the point where only a single kernel instance is present on a compute node in a cluster. Isolated, this bears resemblance to the current approach of the Symmetric Multiprocessing operating systems. However, it is possible to equip two nodes with Popcorn Linux and connect them via the communication layer as well, thus enabling the SSI view over multiple nodes. Most papers and follow-up work regarding Popcorn Linux focus on the usage of the SSI on heterogeneous ISA hardware, for example to utilize both x86 and ARM processors with the same application, since clusters are moving towards heterogeneity. Applying Popcorn Linux to a homogeneous environment is possible as well, even though it is not the main focus of the project.

4 Evaluation

4.1 Prerequisites

To combat the scalability problem of OpenMP described in the introduction, a Single System Image operating system must exhibit at least the following properties: since distributed shared memory is not implemented in current HPC hardware, the operating system has to provide DSM. While less efficient than hardware support, it is a necessity for functioning OpenMP applications, as they rely on shared memory. The second property is thread migration. OpenMP threads need to be either migrated from the initial local node onto remote nodes, or they need to be created at the remote nodes to begin with. Either way the SSI needs to enable such levels of thread management.

In the past, these prerequisites were not necessarily part of every SSI, MOSIX for example did not support programs utilizing shared memory or threads [20]. Kerrighed on the other hand supplied both a software-based DSM mechanism and a thread management mechanism that allowed for migration of threads to remote nodes, or to initially create them on a remote node. This demonstrates that even during the height of the SSI operating systems, not every one of them was fit for the solution of the OpenMP scalability problem that is the subject of this evaluation.

Popcorn Linux also offers software-based DSM, which, alongside the partially replicated OS state between kernels, enables task migration across node boundaries [7]. This encompasses thread migration as well. Both the DSM engine and the task migration make use of the communication layer to communicate updates to memory or to communicate thread state information. While DSM is provided transparently, the migration of threads has to be made explicit. In order for a thread to migrate, a call to the *migrate* function has to be inserted into the user application, once to move the thread onto a remote node and once again at the end to return the thread to the host, returning with the thread state information attached to said thread. The thread migration also requires an extra library that is part of the Popcorn compiler framework. While this compiler framework is built to support the heterogeneous-ISA use case of Popcorn Linux, upon request the developers provided a thread migration library, detached from the extensive compiler framework, for the thread migration in a homogeneous-ISA environment.

4.2 Setup

For the evaluation, the Popcorn Linux kernel version 5.2.21 was used. It was installed on two nodes of the WR cluster at Universität Hamburg, a cluster for educational and research purposes. The nodes in question are equipped with an AMD Opteron 6344 each. The communication layer was established over TCP/IP. The compilation of the kernel itself was carried out with *gcc* v7.5.0. The kernels were installed directly on the hardware instead of a virtual machine to resemble an actual production environment as closely as possible.

4.3 Evaluation

The first part of this evaluation focuses on the initial setup of Popcorn Linux, including any necessary research as well as the installation process. The second part is concerned with the usage and stability of the kernel, followed by a small-scale comparison to tool-based approaches for the solution of the OpenMP scalability issue outlined in the introduction.

Popcorn Linux is documented very poorly. Since it is built on top of the Linux kernel, usual kernel configuration and installation practices can be applied, there is however very little information on the modifications of the kernel or what to take into account during the setup on the Github README or any links contained therein. The same is true for the usage of the thread migration feature. Information on what to consider when trying to migrate threads in a user application is very sparse once again, and the aforementioned library for thread migration in the homogeneous configuration is not available without contacting the developers.

After the initial setup of the nodes, they operate in an isolated manner until the communication layer is established. To that end, a kernel module is provided. After creating a configuration file and loading the kernel module on both nodes, a handshake occurs over TCP/IP, creating the messages displayed in Listing 4.1 on the kernel ring buffer.

```
Loading Popcorn messaging layer over TCP/IP...
popcorn: Loading node configuration...
popcorn: * 0: 10.0.0.63
popcorn:   1: 10.0.0.64
popcorn:    1 joined, x86_64
popcorn: Ready on TCP/IP
```

Listing 4.1: Handshake between two nodes, enabling subsequent communication over the communication layer. The asterisk marks the local node, the other address belongs to the remote node.

After initializing the communication layer, tests were conducted that included communication between the nodes. The first test entails the execution of the command *cat /proc/cpuinfo* in the shell. When the Popcorn communication layer is in place, a call to print the CPU information on one node first prints the information for the local node, which is then followed by the information from the remote node. During the tests, executing said shell command on either node led to the remote node crashing and rebooting, crashing the communication layer in the process, while the local node also froze the ssh session. The same behaviour was observable on any OpenMP application that utilized the Popcorn thread migration. Even simple OpenMP applications that only wrote the hostname to a file (compiled with either *gcc* or *clang*), encapsulated in a *parallel* and a *critical* directive, crashed the remote node while freezing the local one. The crashing node also did not create any log messages regarding the crash, which complicates any attempt at debugging. Upon request, a former developer of the Popcorn Linux kernel tried to assist in finding the cause for the crashes, however to no avail. Due to this, any evaluations of performance when utilizing Popcorn Linux with OpenMP were not conducted.

The instability of Popcorn Linux is not the only downside to the kernel. In its current state, Popcorn Linux can only be used on 32 nodes at most, which is too small for almost any cluster that is currently in use. This emphasizes that the Popcorn Linux kernel, at least in its current state, is experimental and fit for research purposes, but not for a production environment.

4.4 Comparison to Tool-based Approaches

The need to install a custom kernel to utilize the sought-after features of Popcorn Linux, namely the DSM and the thread migration, renders this kernel-based solution undesirable to most operators of HPC systems, as it incurs a loss of flexibility. When compared to the utilization of a tool for automatic transformation of OpenMP applications, the initial cost for the setup is much smaller, as it only requires the installation of the tool, alongside for example a compiler toolchain or other dependencies where necessary.

The instability of Popcorn Linux also points to another crucial downside of the kernel-based approach. If the kernel is unstable to the point where a crash of the communication layer incurs a crash of the entire compute node that the kernel resides on, the administrative overhead for the HPC system operators is increased, whereas a crash of a tool is very unlikely to cause such an outage.

Another downside of Popcorn Linux in particular is that it does not support shared libraries. Applications built for thread migration need to have all their libraries linked in statically. So solving the scalability issue of OpenMP via a special kernel would introduce new restrictions that were not in place beforehand. This pattern extends to most SSIs of the past. For example, MOSIX did not support shared memory or threads and also introduced expensive overhead to several system calls [1]. Introducing limitations to existing working solutions that also affect applications outside of the OpenMP context is not desirable and can be circumvented by using a tool-based solution instead of a kernel. Such a tool-based solution would be able to operate independently of other system components and applications, not introducing any restrictions to otherwise working and independent software.

The final downside is scalability. In the past, SSIs have shown to not scale well beyond low node counts in general, dropping considerably in performance when used on too many nodes [23]. The scalability of the code generated by the tools cannot be generalized, as it would be subject to the specific transformation. If for example a transformation to MPI code is executed, the generated code would be bounded by the scalability of MPI at the very least, not accounting for any further limits imposed by the transformation mechanism itself.

5 Conclusion

In this work, the problem that OpenMP does not scale beyond one compute note currently due to its reliance on shared memory and threads was examined once again, and a solution was evaluated, the Single System Image operating systems. In their current state, SSI operating systems and kernels are not fit to bring OpenMP to multiple nodes and distributed memory. The market for SSIs has dropped, leaving only very few of these operating systems, of which even fewer are focused on HPC clusters. In this work, the evaluation was focused on Popcorn Linux, one of the only remaining

SSI kernels in active development for the HPC market. Due to its poor documentation and instability, Popcorn Linux is not recommended for the use in a homogeneous-ISA production environment, if the reason for its use is to tackle the scalability issue of OpenMP.

Alongside the evaluation of Popcorn Linux, multiple reasons were provided for why the solution to this problem will most likely not be found in the realm of SSI kernels or operating system, ranging from a loss of flexibility to unwanted side effects on otherwise uninvolved software. Instead, a tool-based approach is recommended and these tools should be the focus of further research regarding this problem.

Bibliography

[1] *MOSIX Administrator, User and Programmer Guides and Manuals*, 2022. `https://mosix.cs.huji.ac.il/pub/Guide.pdf`.

[2] George Almasi. *PGAS (Partitioned Global Address Space) Languages*, pages 1539–1545. Springer US, Boston, MA, 2011.

[3] Yariv Aridor, Michael Factor, and Avi Teperman. cJVM: a Single System Image of a JVM on a Cluster. In *Proceedings of the 1999 International Conference on Parallel Processing*, pages 4–11. IEEE, 1999.

[4] Amnon Barak and Oren La'adan. The MOSIX Multicomputer Operating System for High Performance Cluster Computing. *Future Generation Computer Systems*, 13(4-5):361–372, 1998.

[5] Antonio Barbalace, Alastair Murray, Rob Lyerly, and Binoy Ravindran. Towards Operating System Support for Heterogeneous-ISA Platforms. In *Proceedings of The 4th Workshop on Systems for Future Multicore Architectures*, 2014.

[6] Antonio Barbalace, Binoy Ravindran, and David Katz. Popcorn: a replicated-kernel OS based on Linux. In *Proceedings of the Linux Symposium, Ottawa, Canada*, 2014.

[7] Antonio Barbalace, Marina Sadini, Saif Ansary, Christopher Jelesnianski, Akshay Ravichandran, Cagil Kendir, Alastair Murray, and Binoy Ravindran. Popcorn: Bridging the Programmability Gap in Heterogeneous-ISA Platforms. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 1–16, 2015.

[8] Ayon Basumallik and Rudolf Eigenmann. Towards automatic translation of OpenMP to MPI. In *Proceedings of the 19th annual international conference on Supercomputing*, pages 189–198, 2005.

[9] OpenMP Architecture Review Board. *OpenMP Application Programming Interface version 5.2*, 2021. `https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf`.

[10] Rajkumar Buyya, Toni Cortes, and Hai Jin. Single system image. *The International Journal of High Performance Computing Applications*, 15(2):124–135, 2001.

[11] David R Cheriton, Hendrik A. Goosen, and Patrick D. Boyle. Paradigm: A highly scalable shared-memory multicomputer architecture. *Computer*, 24(2):33–46, 1991.

[12] Marie Yvette B. de Robles, Zenith O. Arnejo, and Jaderick P. Pabico. On Web-grid Implementation Using Single System Image. *CoRR*, abs/1507.01067, 2015.

[13] Message Passing Interface Forum. *MPI 3.1 documents*, 2015. `https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf`.

[14] Khaled Hamidouche, Joel Falcou, and Daniel Etiemble. A framework for an automatic hybrid MPI+ OpenMP code generation. In *SpringSim (hpc)*, pages 48–55. Citeseer, 2011.

[15] Philip Healy, Theodore Lynn, Enda Barrett, and John Morrison. Single System Image: A Survey. *Journal of Parallel and Distributed Computing*, 90-91, 02 2016.

[16] Yousef YA Khalidi, Jose M Bernabeu-Auban, Vlada Matena, Ken Shirriff, and Moti Thadani. Solaris MC: A Multi Computer OS. In *USENIX Annual Technical Conference*, pages 191–204, 1996.

[17] Okwan Kwon, Fahed Jubair, Rudolf Eigenmann, and Samuel Midkiff. A hybrid approach of OpenMP for clusters. *ACM SIGPLAN Notices*, 47(8):75–84, 2012.

[18] Chris Lattner and Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, Mar 2004.

[19] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, W-D Weber, Anoop Gupta, John Hennessy, Mark Horowitz, and Monica S. Lam. The Stanford Dash Multiprocessor. *Computer*, 25(3):63–79, 1992.

[20] Renaud Lottiaux, Pascal Gallard, Geoffroy Vallée, Christine Morin, and Benoit Boissinot. OpenMosix, OpenSSI and Kerrighed: A Comparative Study. In *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.*, volume 2, pages 1016–1023. IEEE, 2005.

[21] Christine Morin, Renaud Lottiaux, Geoffroy Vallée, Pascal Gallard, Gaël Utard, Ramamurthy Badrinath, and Louis Rilling. Kerrighed: A Single System Image Cluster Operating System for High Performance Computing. In *European Conference on Parallel Processing*, pages 1291–1294. Springer, 2003.

[22] Rob Pike, Dave Presotto, Sean Dorward, Bob Flandrena, Ken Thompson, Howard Trickey, and Phil Winterbottom. Plan 9 from Bell Labs. *Computing systems*, 8(3):221–254, 1995.

[23] Rolf Riesen and Arthur B. Maccabe. *Single System Image*, pages 1820–1827. Springer US, Boston, MA, 2011.

[24] Jannek Squar, Tim Jammer, Michael Blesel, Michael Kuhn, and Thomas Ludwig. Compiler Assisted Source Transformation of OpenMP Kernels. In *2020 19th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 44–51. IEEE, 2020.

[25] Bruce Walker, Gerald Popek, Robert English, Charles Kline, and Greg Thiel. The LOCUS Distributed Operating System. *ACM SIGOPS Operating Systems Review*, 17(5):49–70, 1983.