Implementing Transparent Shared Memory on Clusters Using Virtual Machines

Matthew Chapman and Gernot Heiser
The University of New South Wales, Sydney, Australia
National ICT Australia, Sydney, Australia
matthewc@cse.unsw.edu.au

Abstract

Shared memory systems, such as SMP and ccNUMA topologies, simplify programming and administration. On the other hand, clusters of individual workstations are commonly used due to cost and scalability considerations.

We have developed a virtual-machine-based solution, dubbed vNUMA, that seeks to provide a NUMA-like environment on a commodity cluster, with a single operating system instance and transparent shared memory. In this paper we present the design of vNUMA and some preliminary evaluation.

1 Introduction

Many workloads require more processing power than feasible with a single processor. Shared-memory multiprocessors, such as SMP and NUMA systems, tend to be easier to use, administer and program than networks of workstations. Such shared-memory systems often use a *single system image*, with a single operating system instance presenting a single interface and namespace. On the other hand, clusters of individual workstations tend to be a more cost-effective solution, and are easier to scale and reconfigure.

Various techniques have been proposed to provide the simplicity of shared-memory programming on networks of workstations. Most depend on simulating shared memory in software by using virtual memory paging, known as *distributed shared memory* (DSM) [1]. At the middleware layer there are DSM libraries available, such as Treadmarks [2]. These libraries require software to be explicitly written to utilise them, and they do not provide other facets of a single system image such as transparent thread migration. Some projects have attempted to retrofit distribution into existing operating systems, such as the MOSIX clustering software for Linux [3]. However, Linux was not designed with such distribution in mind, and while MOSIX can provide thread migration, many system calls still need to be routed back to the original node. Other projects have attempted to build distributed operating systems from the ground up, such as Amoeba [4] and Mungi [5]. In order to gain wide acceptance, these operating systems need to provide compatibility with a large body of existing UNIX applications, which is no easy task.

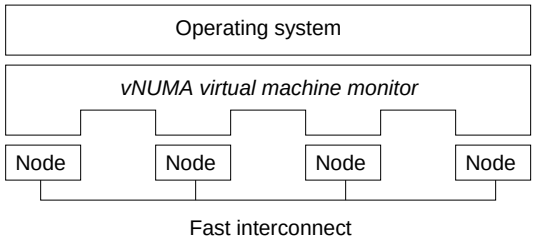

Figure 1: Cluster with vNUMA

In this paper, we present an alternative approach utilising virtualisation techniques. Virtualisation can be useful for hiding hardware complexities from an operating system. A privileged *virtual machine monitor* interposes between the operating system and the hardware, presenting virtual hardware that may be different from the real hardware. For example, Disco [6] simulates multiple virtual SMP systems on a NUMA system.

vNUMA uses virtualisation to do essentially the opposite — simulating a single virtual NUMA machine on multiple workstations, using DSM techniques to provide shared memory. Unlike previous work, this can achieve a true single system image using a legacy operating system, without significant modifications to that operating system.

We focus on Linux as the guest operating system, since it supports NUMA hardware and the source code is available. This means that there are already some optimisations to improve locality, and we can make further improvements if necessary.

We chose to target the Itanium architecture [7] for our virtual machine. Numerous IA-32 virtual machine monitors already exist, and a number of the techniques are encumbered by patents. Itanium is being positioned by Intel as the next "industry standard architecture", particularly for high-end systems. An Itanium virtual machine monitor presents some research opportunities in itself, independent of the distribution aspects.

First published in *Proceedings of USENIX 05: General Track* Anaheim, CA, USA, April 2005.

2 Implementation overview

2.1 Startup

In order to achieve the best possible performance, vNUMA is a *type I* VMM; that is, it executes at the lowest system software level, without the support of an operating system. It is started directly from the bootloader, initialises devices and installs its own set of exception handlers.

One of the nodes in the cluster is selected as the bootstrap node, by providing it with a guest kernel as part of the bootloader configuration. When the bootstrap node starts, it relocates the kernel into the virtual machine's address space, and branches to the start address; all further interaction with the virtual machine is via exceptions. The other nodes wait until the startup node provides a start address, then they too branch to the guest kernel; its code and data is fetched lazily via the DSM.

2.2 Privileged instruction emulation

In order to ensure that the virtual machine cannot be bypassed, the guest operating system is demoted to an unprivileged privilege level. Privileged instructions then fault to the virtual machine monitor. The VMM must read the current instruction from memory, decode it, and emulate its effects with respect to the virtual machine. For example, if the instruction at the instruction pointer is `mov r16=psr`, the simulated PSR register is copied into the userspace `r16` register. The instruction pointer is then incremented.

The Itanium architecture is not perfectly virtualisable in this way and has a number of sensitive instructions, which do not fault but require VMM intervention [8, 9]. These must be substituted with faulting instructions. Currently this is done statically at compilation time, although it would be possible to do at runtime if necessary, since the replacement instructions are chosen so that they fit into the original instruction slots. The `cover` instruction is simply replaced by `break`. `thash` and `ttag` are replaced by moves from and to *model-specific registers* (since model-specific registers should not normally be used by the operating system, and these instructions conveniently take two register operands).

2.3 Distributed Shared Memory

The virtual machine itself has a simulated physical address space, referred to here as the machine address space. This is the level at which DSM operates in vNUMA. Each machine page has associated protection bits and other metadata maintained by the DSM system. When the guest OS establishes a virtual mapping, the effective protection bits on the virtual mapping are calculated as the logical AND of the requested protection bits and the DSM protection bits. vNUMA keeps track of the virtual mappings for each machine page, such that when the protection bits are updated by the DSM system, any virtual mappings are updated as well.

The initial DSM algorithm is a simple sequentially consistent, multiple-reader/single-writer algorithm, based on that used in IVY [10] and other systems. The machine pages of the virtual machine are divided between the nodes, such that each node manages a subset of the pages. When a node faults on a page, the manager node is contacted in the first instance. The manager node then forwards to the owner (if it is not itself the owner), and the owner returns the data directly to the requesting node. The copyset is sent along with the data, and if necessary the receiving node performs any invalidations. Version numbers are used to avoid re-sending unchanged page data.

3 Evaluation

Our test environment consists of two single-processor 733Mhz Itanium 1 workstations with D-Link DGE-500T Gigabit Ethernet cards, connected back-to-back with a crossover cable to form a two-processor cluster. We also used a similar dual-processor (SMP) Itanium workstation for comparison. Obviously it is intended that the system will scale beyond two nodes, however the software was not yet stable enough for benchmarking on a larger cluster.

As the guest kernel, we used a Linux 2.6.7 kernel compiled for the HP simulator platform. The only modifications to the kernel are a tiny change to enable SMP (since the HP simulator is usually uniprocessor), and the static instruction replacement described in section 2.2.

The SPLASH-2 benchmarks [11] are a well-known set of benchmarks for shared memory machines. We used an existing implementation designed to work with the standard *pthreads* threading library. Here we present results from three of the SPLASH-2 applications: **Ocean**, **Water-Nsquared** and **Barnes**. In each case we measured the performance on four different topologies: a single-processor workstation, a single-processor workstation with vNUMA (to measure virtual machine overhead), two single-processor workstations with vNUMA, and a dual-processor SMP workstation. We used the processor cycle counter to obtain timings, since we did not want to place trust in the accuracy of `gettimeofday` on the virtual machine.

3.1 Ocean

Ocean simulates large-scale ocean movements by solving partial diferential equations. The grid representing the ocean is partitioned between processors. At each iteration the computation performed on each element of the grid requires the values of its four neighbours, causing communication at partition boundaries.

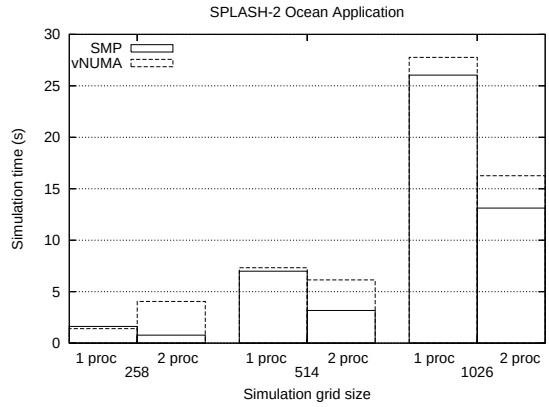

Figure 2: Results of **Ocean** application

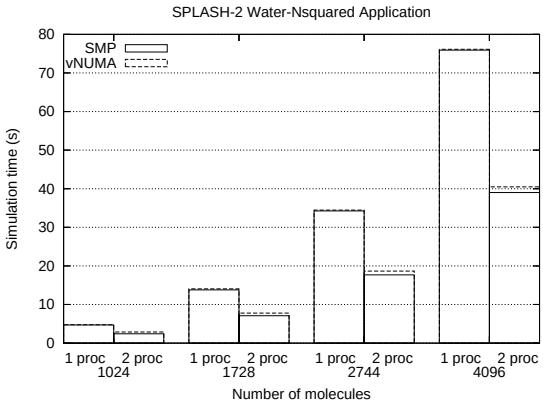

Figure 3: Results of **Water-Nsquared** application

The results are shown in Figure 2. First consider the single processor results, which demonstrate virtual machine performance independent of the DSM. At the smallest grid size, 258x258, the virtual machine performance is very good, in fact the benchmark runs marginally faster than without the virtual machine. This is due to the fact that parts of the memory management are done by the virtual machine monitor without involving the guest operating system, and the mechanisms implemented in vNUMA (such as the long format VHPT) are advantageous for some workloads compared to those implemented in Linux [12]. As the grid size and hence working set size increases, the number of TLB misses and page faults that must involve the guest kernel increases. Since these are significantly more expensive on the virtual machine, they ultimately outweigh any memory management improvements. At the largest grid size the virtual machine imposes a 7% overhead.

On the other hand, the distribution efficiency increases with problem size. If the granularity was word-based, communication should increase linearly with one side of the grid. However, because of sparse access patterns compared to the granularity of the DSM, we simply see greater utilisation of the pages being transferred, and the overhead remains roughly constant, meaning that the relative overhead is less. For the 258x258 grid, the vNUMA overhead is significant compared to the actual amount of work being done, and it is clearly not worthwhile. By 514x514, we have passed the "break-even" point and the two-node vNUMA performs better than a single processor. For the largest problem size, the benchmark is largely computation-bound and vNUMA works well. Relative to a single-processor workstation, the vNUMA speedup is 1.60, compared to 1.98 for SMP.

3.2 Water-Nsquared

Water-Nsquared is an example of an application that performs well in a DSM environment [13], and indeed it also performs well on vNUMA. **Water-Nsquared** evaluates forces and potentials that occur over time in a system of water molecules. Each processor needs all of the data, but only does a subset of the calculations and stores the results locally. At the end of each timestep, processors accumulate their results into the shared copy. Thus there are alternating read-sharing and update phases.

The results are shown in Figure 3. Here the virtual machine overhead is minimal, since the working set sizes are much smaller than for **Ocean** (around 4MB at the largest problem size, compared to over 220MB). The distribution overhead scales with the number of molecules (and hence the size of the shared data), as might be expected, but again it is small. For the largest problem size, the vNUMA speedup is 1.87, compared to 1.95 for SMP.

3.3 Barnes

On the other hand, **Barnes** is an example of an application that is known not to perform as well in DSM environments [13]. **Barnes** simulates the gravitational interaction of a system of bodies in three dimensions using the Barnes-Hut hierarchical N-body method. The data is represented as an octree with leaves containing information on each body and internal nodes representing space cells. Thus there are two stages in each timestep — calculating forces and updating particle positions in the octree.

The results are shown in Figure 4. The force calculation phase distributes fairly well, certainly for larger problem sizes. However the tree update does not — in this phase the pattern of both reads and writes is fine-grained and unpredictable, which results in significant false sharing. False sharing is particularly problematic because vNUMA currently uses a sequentially consistent, multiple-reader/single-writer DSM, which means pages cannot simultaneously be writable on multiple nodes. Thus, overall, the benchmark does not perform

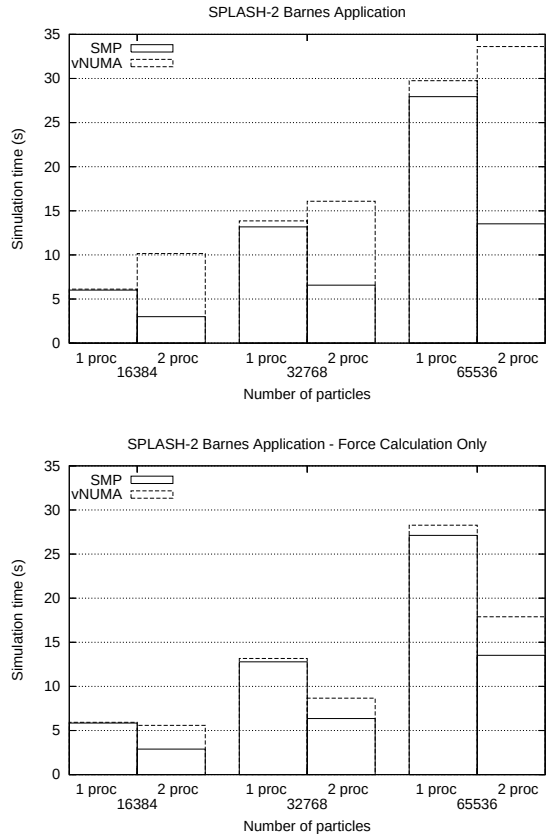

Figure 4: Results of **Barnes** application

well on vNUMA.

4 Conclusions

These results show that, at least for scientific applications such as those in the SPLASH-2 suite, vNUMA performance can be surprisingly good and is dominated by application DSM costs rather than virtualisation or kernel paging overheads. Applications that behave well on conventional DSM systems, such as **Water-Nsquared**, perform best on vNUMA. These are typically applications which are computation-intensive and share pages mostly for reading rather than writing.

However vNUMA has significant advantages over middleware DSM systems, providing a true single system image and a simple migration path for SMP applications. Since it utilises networks of commodity workstations, it is more cost-effective and reconfigurable than specialised ccNUMA hardware. We believe that, at least for some classes of applications, vNUMA could provide a useful alternative to these systems. There are still improvements to be made, and we need to perform benchmarks on larger clusters to prove scalability.

5 Acknowledgements

This work was supported by a Linkage Grant from the Australian Research Council (ARC) and a grant from HP Company via the Gelato.org project, as well as hardware from HP and Intel. National ICT Australia is funded by the Australian Government's Department of Communications, Information Technology and the Arts and the ARC through *Backing Australia's Ability* and the ICT Research Centre of Excellence programs.

References

[1] K. Li. *Shared Virtual Memory on Loosely Coupled Multiprocessors*. Phd thesis, Yale Univ., Dept. of Computer Science, 1986. RR-492.

[2] P. Keleher, S. Dwarkadas, A. L. Cox, W. Zwaenepoel. Treadmarks: Distributed shared memory on standard workstations and operating systems. In *Proc. of the Winter 1994 USENIX Conference*, p 115–131, 1994.

[3] A. Barak, O. La'adan, A. Shiloh. Scalable cluster computing with MOSIX for Linux. In *Proceedings of the 5th Annual Linux Expo*, p 95–100, 1999.

[4] S. J. Mullender, G. van Rossum, A. S. Tanenbaum, R. van Renesse, H. van Staveren. Amoeba: A distributed operating system for the 1990s. *IEEE Computer*, 23(5):44–53, 1990.

[5] G. Heiser, K. Elphinstone, J. Vochteloo, S. Russell, J. Liedtke. The Mungi single-address-space operating system. *Softw.: Pract. & Exp.*, 28(9):901–928, Jul 1998.

[6] E. Bugnion, S. Devine, M. Rosenblum. Disco: Running commodity operating systems on scalable multiprocessors. In *Proc. 16th SOSP*, p 27–37, 1997.

[7] Intel Corp. *Itanium Architecture Software Developer's Manual*, Oct 2002. `http://developer.intel.com/design/itanium/family/`.

[8] C. Gray, M. Chapman, P. Chubb, D. Mosberger-Tang, G. Heiser. Itanium —a system implementor's tale. In *Proc. 2005 USENIX Techn. Conf.*, Anaheim, CA, USA, Apr 2005.

[9] D. J. Magenheimer, T. W. Christian. vBlades: Optimised paravirtualisation for the Itanium processor family. In *Proc. 3rd Virtual Machine Research & Technology Symp.*, p 73–82, 2004.

[10] K. Li, P. Hudak. Memory coherence in shared virtual memory systems. *Trans. Comp. Syst.*, 7:321–59, 1989.

[11] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proc. 22nd ISCA*, p 24–36, 1995.

[12] M. Chapman, I. Wienand, G. Heiser. Itanium page tables and TLB. Technical Report UNSW-CSE-TR-0307, School Comp. Sci. & Engin., University NSW, Sydney 2052, Australia, May 2003.

[13] L. Iftode. *Home-based Shared Virtual Memory*. PhD thesis, Princeton University, Dept of Computer Science, 1998.