

Data Distribution, Migration and Replication on a cc-NUMA Architecture

J. Mark Bull and Chris Johnson
EPCC, The King's Buildings, The University of Edinburgh,
Mayfield Road, Edinburgh EH9 3JZ, Scotland, U.K.
email: [m.bull, c.johnson]@epcc.ed.ac.uk

1 Introduction

It is well known that, although cc-NUMA architectures allow construction of large scale shared memory systems, they are more difficult to program effectively because data locality is an important consideration. Support for specifying data distribution in OpenMP has been the subject of much debate [1], [4], and several proposed implementations. These take the form of data distribution directives, giving the programmer control of where data is placed in the memory system. In the absence of additional directives, data distribution can be controlled by exploiting the system's allocation policy. In most cc-NUMA systems, data is placed on the node which first accesses it: the so-called *first touch* policy. An alternative strategy is to give this control not to the programmer but to the operating system, by allowing the location of data in memory to change as a program executes. This can be done either by data migration, where pages can move between nodes, but there is only ever one copy, or by replication, where multiple copies of pages can exist.

In this study, we examine the interactions between data distribution (implemented via the first touch policy), migration, and replication on a prototype cc-NUMA architecture. On this system it appears that replication is almost always more effective than migration, despite the additional cost in memory usage. Data distribution can be effective in applications where there is obviously a "correct" distribution. In many applications the "correct" distribution is not so obvious, and although it pays to distribute data in the absence of migration and replication, some combination of replication and migration can often achieve comparable performance.

2 The Sun Wildfire prototype

The Sun Wildfire system [3], [5] is a prototype cc-NUMA architecture, built from standard SMP nodes. In each SMP, one processor board is replaced by a Wildfire interconnect board, which can have up to three high-speed links to other SMP nodes, allowing construction of machine with up to four nodes. Unlike other cc-NUMA machines (such as the SGI Origin series) the Wildfire system has a small number of (potentially) large nodes, rather than a large number of small nodes.

The Wildfire system at the University of Edinburgh consists of three nodes: one E6000 server with 18 processors and two E4000 servers with 8 processors each. The processors are 250MHz Ultrasparc IIs, and the system runs a modified version of Solaris 6. The memory latency for a memory access to a remote node is around 6-7 times that for an access to main memory on the local node. This is quite a large factor for a cc-NUMA architecture.

Page migration and replication are managed by daemons running on each node, and can be switched on and off on a system-wide basis via a command line interface. The algorithms for determining when a page should be migrated or replicated are described in [5]. The system monitors remote accesses to pages, and when these exceed a certain threshold, the page is marked as a candidate for replication or

migration. If both are enabled, in general migration is tried first, and a page is replicated if it is already been replicated, or has been recently migrated. When pages are replicated, a “shadow” page is set up on the local node, which can satisfy misses for the data on that page. Cache coherency is still maintained at the cache-line level on a system-wide basis.

3 Experiments and Results

To evaluate the interactions between distribution, migration and replication, we have taken a simple two-dimensional CFD simulation, and number of codes from the OpenMP version of the NAS Parallel Benchmark suite [2]. Two versions of each code were produced: with sequential and parallel data initialisation to control data distribution. In the case of the NAS benchmarks, the supplied code usually contains parallelised data initialisation, so to obtain a sequential initialisation we removed the relevant OpenMP directives. Each version was run with migration and replication independently enabled and disabled, giving a total of eight runs for each code. The codes were run on 18 threads, utilising six processors on each of the three nodes. OpenMP threads were bound to processors to prevent threads being migrated between nodes. Experiments with different number of processors suggest that similar conclusions would be drawn, and so for sake of clarity we do not report them here.

	Distribution OFF		Distribution ON	
	Migration OFF	Migration ON	Migration OFF	Migration ON
Replication OFF	275	79	63	63
Replication ON	78	81	63	63

Table 1: Execution time (in seconds) of SHALLOW on 18 processors

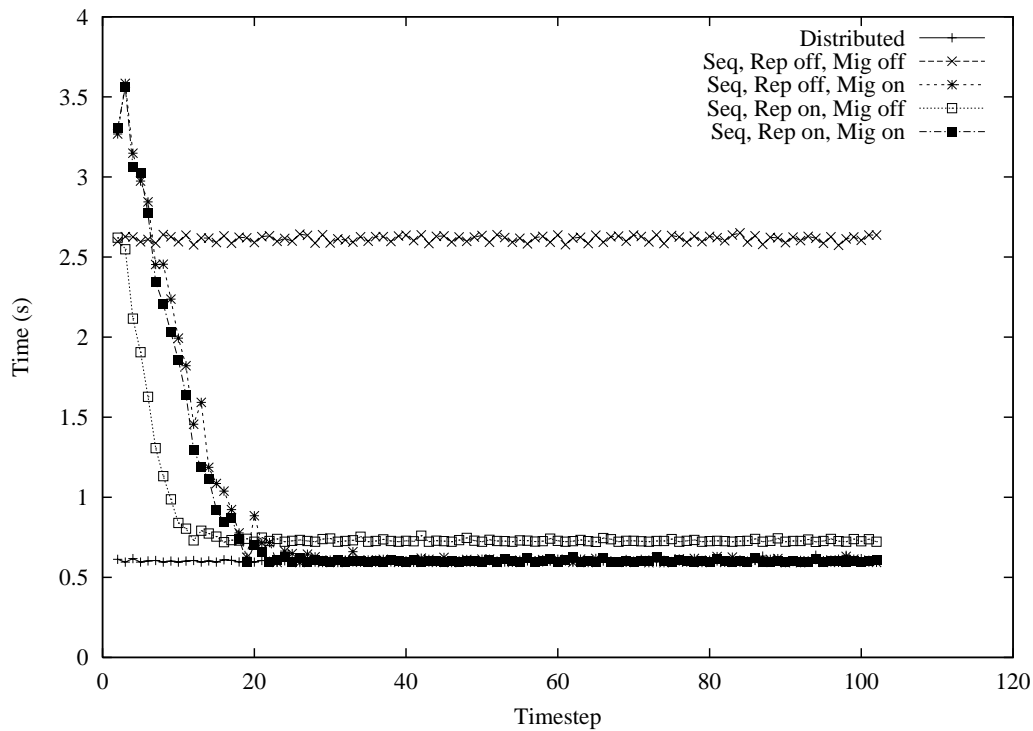


Figure 1: Execution time (in seconds) per timestep for SHALLOW on 18 processors

	Distribution OFF		Distribution ON	
	Migration OFF	Migration ON	Migration OFF	Migration ON
Replication OFF	26.0	5.94	6.11	6.12
Replication ON	7.25	6.21	6.09	6.10

Table 2: Execution time (in seconds) of final 10 timesteps of SHALLOW on 18 processors

Table 1 shows the execution time for SHALLOW, a simple 2-D shallow water simulation, on the Wildfire system. Without data distribution, all memory is allocated on one node. By distributing the data, a more than four-fold performance increase is obtained, and replication and migration have no additional benefit. Without distribution, migration, replication and the combination of the two all reduce the run time significantly, to within a factor of 1.4 of that achieved by distribution. As the run time is quite short, transient effects are still significant: these are observable in Figure 1, in which the execution time for each of the 100 timesteps is displayed. Transient behaviour is seen in the first 20 or 25 timesteps.

Table 2 shows the execution time of the last 10 timesteps out of the 100 executed in this run of the code. Migration alone achieves a slightly better performance than distribution: replication in addition to migration does not help here, and replication alone is around 20% slower than distribution.

As an aside, it is interesting to note that if we run the same simulation on the 18 processors of the E6000, then the execution time is approximately 10% *longer* than when distributed across the three nodes. In the latter case, the additional memory bandwidth available outweighs any penalty of longer latencies across the Wildfire interconnect.

	Distribution OFF		Distribution ON	
	Migration OFF	Migration ON	Migration OFF	Migration ON
Replication OFF	965	609	620	601
Replication ON	588	581	582	578

Table 3: Execution time (in seconds) of BT, Class B, on 18 processors

Table 3 shows the execution time for BT from the NAS suite. In this case data access patterns are more complex than in SHALLOW, and the best data distribution strategy is less obvious. We observe that without migration or replication, data distribution has a significant effect on performance. However, both migration and replication, and in particular the combination of both, achieve better performance than distribution alone. Indeed, if at least one of these is enabled, data distribution has very little additional benefit.

	Distribution OFF		Distribution ON	
	Migration OFF	Migration ON	Migration OFF	Migration ON
Replication OFF	1065	699	941	699
Replication ON	296	279	298	288

Table 4: Execution time (in seconds) of CG, Class B, on 18 processors

Table 4 shows the execution time for CG from the NAS suite. In this case the distribution of pages by first touch policy is not very effective, reducing the execution time from 1065 seconds to 941 seconds. Migration is able to reduce the time to 699 seconds, but replication is by far the most effective strategy, regardless of whether migration or distribution is used.

Table 5 shows the execution time for FT from the NAS suite. All three strategies are beneficial, but the best performance is obtained by combining all three. This is the only case in our experiments where

	Distribution OFF		Distribution ON	
	Migration OFF	Migration ON	Migration OFF	Migration ON
Replication OFF	516	328	375	264
Replication ON	249	261	187	198

Table 5: Execution time (in seconds) of FT, Class B, on 18 processors

it appears that benefits of distribution cannot be replicated by using migration or replication. However, we have not discounted the possibility that this is a transient effect and that a longer run time would not show such a significant advantage for distribution.

	Distribution OFF		Distribution ON	
	Migration OFF	Migration ON	Migration OFF	Migration ON
Replication OFF	231	230	236	234
Replication ON	220	225	224	228

Table 6: Execution time (in seconds) of MG, Class C, on 18 processors

Table 6 shows the execution time for MG from the NAS suite. In this case neither distribution, replication nor migration has a significant effect on the execution time.

	Distribution OFF		Distribution ON	
	Migration OFF	Migration ON	Migration OFF	Migration ON
Replication OFF	1541	781	764	778
Replication ON	675	680	666	670

Table 7: Execution time (in seconds) of SP, Class B, on 18 processors

Table 7 shows the execution time for SP from the NAS suite. Here we observe a similar situation as for BT: both distribution and migration show a significant benefit, but replication is the best strategy. With replication enabled, it makes little difference whether the other two strategies are used or not.

4 Discussion

The results we present above indicate that, at least for this type of code, the dynamic techniques of replication and migration are able to obtain performance as good as, and sometimes significantly better than, static data distribution. We acknowledge the limitations of this experiments: there is little pressure on memory which might put replication (and to some extent migration) at a disadvantage in cases where a large fraction of the physical memory is required for an application. Furthermore the system we have used is on the small side, though we might expect to obtain similar results on a system with a small number of large SMP nodes. This type of system has so far not found favour commercially: previous distributed shared memory systems have typically been constructed with a large number of small nodes each containing between one and four processors. An advantage of the large-node design is that the scalability of the dynamic techniques is not severely tested.

Another important observation is that replication is, if anything, more beneficial than migration, which has tended to receive more attention in the literature. It has been noted, for example in [4], that migration alone can result in pages which are referenced by multiple nodes bouncing around the system unless some action is taken to prevent this. In the Wildfire system, such pages are no longer migrated, but replicated instead.

Nevertheless, migration is still a useful strategy, so it would be preferable for the user to retain some control over the use of replication and migration. In the Wildfire system such control operates on a system wide basis, which is undesirable if the system is to be used to run multiple applications at the same time.

Our findings therefore tend to support those of [4] who conclude that data distribution is not a necessary extension for OpenMP, because runtime techniques are sufficiently powerful. In the past year, the economic climate has forced a number of vendors to cancel or postpone plans to build large cc-NUMA machines. Instead, there has been a trend towards building larger SMP systems (for example Sun Fire 15000, Fujitsu PrimePower) with better scalability of memory bandwidth, although bandwidth limitations in these systems are still a significant obstacle to the scaling of some applications.

However, it may be possible that interest will be renewed in large cc-NUMA machines in the future. If so, then it will be critical to co-design the hardware and operating system to implement effective dynamic page placement strategies for large scale scientific applications and to give the user adequate control over their use. Given the technical difficulties involved, further research in this area could be a sound investment for the future of large shared memory architectures.

References

- [1] John Bircsak, Peter Craig, RaeLyn Crowell, Zarka Cvetanovic, Jonathan Harris, C. Alexander Nelson, Carl D. Offner, *Extending OpenMP for NUMA Machines*, in Proc. of IEEE/ACM Supercomputing'2000: High Performance Computing and Networking Conference, Dallas TX, November 2000.
- [2] Bailey, D. H., E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrisnan, and S. Weeratunga, *The NAS Parallel Benchmarks* Technical Report RNR-94-007, NASA Ames Research Center, March 1994.
- [3] E. Hagersten and M. Koster *WildFire: A scalable path for SMPs*, in Proceedings of the Fifth IEEE Symposium on High-Performance Computer Architecture, pages 172–181, February 1999.
- [4] D.S. Nikolopoulos, T.S. Papatheodorou, C.D. Polychronopoulos, J. Labarta and E. Ayguade *Is Data Distribution Necessary in OpenMP?*, in Proc. of IEEE/ACM Supercomputing'2000: High Performance Computing and Networking Conference, Dallas TX, November 2000.
- [5] L. Noordergraaf and R. Van der Pas, *Performance Experiences on Sun's Wildfire Prototype*, in Proc. of Supercomputing '99, Portland, OR, November 1999.