Adaptive Proxies: Handling Widely-Shared Data in Shared-Memory Multiprocessors

Sarah A. M. Talbot Paul H. J. Kelly

Department of Computing, Imperial College of Science, Technology and Medicine, 180 Queen's Gate, London SW7 2BZ, United Kingdom

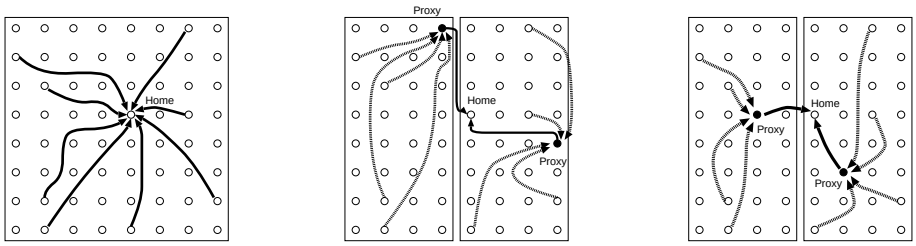
Abstract. A performance bottleneck arises in distributed shared-memory multiprocessors when there are many simultaneous requests for the same data. One architectural solution is to distribute read requests to nodes other than the home node: these other nodes act as intermediaries (*i.e.* proxies) in obtaining the data, and combine requests for the same data. Adaptive proxies use proxying during the proxying period, which varies depending on the level of run-time congestion. Simulation results show that adaptive proxies give performance improvements for all our benchmark applications.

1 Introduction

In a cache-coherent non-uniform access (cc-NUMA) shared-memory multiprocessor, remote access to each processor's memory and local cache is managed by a "node controller". In large configurations, unfortunate ownership migration or home allocations can lead to the concentration of requests for data at particular nodes. This results in the performance being limited by the service rate or "occupancy" of an individual node controller [3].

In this paper we present an adaptive proxy cache coherency protocol, which alleviates contention for widely-shared data, and can do so without adversely affecting any of the applications we have simulated. The adaptive proxy scheme requires no modification or annotation to the application code. The additional protocol complexity and hardware requirements are small: proxying could probably be added to a typical firmware node controller with no hardware change. In our earlier work on proxies, any data obtained by a node acting as a proxy was cached in the processor's second level cache [7]. This was done deliberately to increase the combining effect, *i.e.* further read requests for that data can be satisfied at the proxy. However, the drawbacks include increased sharing list length, cache pollution, and delays to the local processor and node controller processing. The results in this paper include two new caching options: not caching proxy data, and using a separate buffer for proxy data (with access latencies the same as for accessing DRAM).

The rest of the paper is structured as follows: Section 2 introduces adaptive proxies. Our simulated architecture and experimental design are outlined in Section 3. The results of execution-driven simulations for a set of eight benchmark programs are presented in Section 4. Finally, in Section 5, we summarise our conclusions and give pointers to further work.



(a) Without proxies

(b) With two proxy clusters (read data line l)

(c) Read next data line (read data line $l+1$)

Fig. 1. Contention is reduced by routing reads via a proxy

2 Adaptive Proxies

In the proxy scheme, a processor issuing a read request for remote data sends the request message to another node, which is known to act as a proxy for that data line, rather than going directly to the data line's home node [7]. The number of proxy clusters ($\mathcal{NPC}$) is 2 in the example shown in Fig. 1, *i.e.* each processing node has been allocated to one of two sets (this can be done on the basis of network locality). Home node congestion is the run-time trigger for using proxies. In large-scale systems it is impractical to provide enough buffering at each node to hold all the incoming messages, and a commonly adopted strategy handles a `read-request` that reaches a full buffer by sending a negative acknowledgement (a NAK) back to the requester. The results for reactive proxies were encouraging [7], but the scheme suffered from incurring the delay (before the NAK arrives to signal that a proxy read request is needed) each time data is required from a congested home node.

Adaptive proxies use the arrival of a NAK'd `read-request` message to trigger the start of a proxy-period, *i.e.* a time during which any further `read-request` messages destined for the home node are replaced with `proxy-read-request` messages. The proxy-period is modified according to the level of NAKs, using a random walk policy [1]. The probability of a NAK (from a particular home node) occurring within an upper time limit of the last NAK (from that home) is high if the last "inter-NAK" period was less than the upper time limit.

The adaptive proxy policy is controlled by the following parameters: current time T_{curr}, PP_{unit} is one unit of proxy-period time (set to 1000 cycles), PP_{max} is the maximum proxy-period (set to 50), PP_{min} is the minimum proxy-period (set to 1). Each node controller x is extended with two vectors: $LB_{(x,y)}$ gives for each remote node y the time at which the last (NAK) message was received at client node x from node y, and $PP_{(x,y)}$ maintains the current proxy-period for reads by this client x to each remote node y. The arrival at client node x of a NAK from home node y will trigger the adjustment of $PP_{(x,y)}$ as follows:

$$PP_{(x,y)} = \begin{array}{l} min(PP_{max}, PP_{(x,y)}+1) \textbf{ if } (T_{curr} - LB_{(x,y)}) < (PP_{unit} \times PP_{max}) \\ max(PP_{min}, PP_{(x,y)}-1) \textbf{ otherwise} \end{array}$$

The choice of suitable values for PP_{max}, PP_{min}, and PP_{unit} depends on the architecture, and the values used in this paper were selected after experiments

with a range of values. To decide whether proxying is appropriate, there has to be an extra check before each `read-request` is issued by a client x to a home node y:

if $[LB_{(x,y)} > 0]$ **and** $[(PP_{(x,y)} \times PP_{unit}) > (T_{curr} - LB_{(x,y)})]$
then send a `proxy-read-request`,
otherwise send a normal `read-request`.

3 Simulated Architecture and Experimental Design

The cc-NUMA design which is simulated for this work has already been described in [7], so this section concentrates on the changes required to support adaptive proxies and alternative strategies for caching proxied data. The caches are kept coherent using an invalidation-based, distributed directory protocol using singly-linked lists [9]. The benchmark applications are summarised in Table 1. GE implements a Gaussian Elimination algorithm [2]. CFD is a computational fluid dynamics application modelling laminar flow [8]. The remaining six applications were taken from Stanford's SPLASH-2 suite [10].

The adaptive proxies scheme adjusts the proxying period according to the level of congestion at individual home nodes. However it has the storage overheads of holding the $LB_{(x,y)}$, $PP_{(x,y)}$, PP_{unit}, PP_{max}, and PP_{min} values at each node. There are also the processing overheads of adjusting $PP_{(x,y)}$, and checking before issuing each remote read-request.

Implementing a separate proxy buffer would require a node controller which is capable of using a small area of the local memory for its own purposes (*e.g.* [4]), or which has some dedicated storage within the node controller (similar to [5]).

4 Experimental Results

This section presents the results obtained from execution-driven simulations of the adaptive proxy strategy using the three proxy data caching policies[1]. The results are summarised in Table 2, and are presented in terms of relative speedup, *i.e.* the ratio of the execution time for the fastest algorithm running on one processor to the execution time on $\mathcal{P}$ processors. For proxy caching in the SLC, the read-requests benefit from being spread around the system during the proxying period. However the scheme suffers from over-using proxies for the Ocean-Contig application (cache pollution and too large a value for PP_{unit}), and so has no overall balance point for the eight benchmark applications[2]. The GE application exhibits some bottleneck problems when $\mathcal{NPC}$=1&2, where proxy messages are sent to an already congested node, leading to a rise in overall queueing delay (although this is compensated for by the gains at other nodes).

The non-caching proxy policy results show that the proxying technique is still effective even when the opportunities for combining are restricted. The balance point at $\mathcal{NPC}$=1 occurs both because the chances of combining are greatest

[1] A detailed analysis of the simulation results can be found in [6].

[2] A balance point is where the partition into $\mathcal{NPC}$ proxy clusters results in improved performance for all eight benchmark applications.

Table 1. Benchmark applications

application	problem size	application	problem size
Barnes	16K particles	GE	512 × 512 matrix
CFD	64 × 64 grid	Ocean-Contig	258 × 258 ocean
FFT	64K points	Ocean-Non-Contig	258 × 258 ocean
FMM	8K particles	Water-Nsq	512 molecules

(since there is only one proxy node for a given data line), and because the Ocean-Contig application is able to benefit from the reduced cache pollution.

With a separate proxy buffer, there are three balance points, at $\mathcal{NPC}$=2,6,&7. The proxy buffer technique avoids the cache interference patterns seen with SLC caching for Barnes and Ocean-Non-Contig, while keeping most of the benefits of combining (unlike the non-caching approach). Ocean-Non-Contig in particular, which has poor data locality, benefits from the reduction in SLC cache pollution and the combining of proxy read requests. The results for Ocean-Contig highlight a subtle side-effect of using proxies. For values of $\mathcal{NPC} \geq 1$ the performance is determined by the effect of the use of proxies on the overall barrier delay. The changes in barrier delay result from redistributing messages to proxy nodes and the delays experienced by other messages queueing for service at proxy nodes.

Overall the adaptive proxy scheme gets the best performance with the separate proxy buffer, obtaining balance points at $\mathcal{NPC}$=2,6&7. A balance point is more desirable than a value of $\mathcal{NPC}$ which gives the best result for a specific application because we aim to get reasonable performance for a wide range of applications without the need to tune applications to suit the system. However, the no-proxy-caching strategy (when $\mathcal{NPC}$=1 to maximise combining) is a reasonable solution where it is not possible to have proxy buffers.

5 Conclusions and Further Work

This paper has proposed adaptive proxies to alleviate the performance problems arising from read accesses to widely-shared data. The simulation results show that adaptive proxies (with a separate proxy buffer or with no-cache-proxies) give stable performance, allow the programmer to write portable applications which are less "architecture specific", and save on performance tuning because the widely-shared data access bottleneck is dealt with automatically. To evaluate the commercial viability of adaptive proxies it would be necessary to investigate the performance effects of commercial workloads. Further work is also needed to assess the impact of different network topologies and processor cluster nodes, and alternative implementations of the proxy buffer.

Acknowledgements

This work was funded by the U.K. Engineering and Physical Sciences Research Council through a Research Studentship. We would also like to thank Ashley Saulsbury and Andrew Bennett for their work on the ALITE simulator.

Table 2. Benchmark relative speedups with a separate proxy buffer (64 processors)

application	relative speedup no proxies	proxy caching method	% change in execution time (+ is better, − is worse) for $\mathcal{NPC}$ = 1 to 8							
			1	2	3	4	5	6	7	8
Barnes	46.3	SLC	+0.1	+3.2	+0.4	+0.4	+0.4	+0.2	-0.1	+0.2
		none	+0.4	+3.7	0.0	0.0	+0.5	+0.3	+0.1	+0.2
		buffer	0.0	+3.3	+0.4	+0.2	+0.2	+0.2	+0.4	+0.4
CFD	28.3	SLC	+9.2	+13.1	+11.3	+11.6	+11.2	+10.4	+10.6	+12.1
		none	+12.9	+13.7	+13.6	+12.7	+12.9	+13.5	+12.7	+12.5
		buffer	+9.4	+9.4	+9.0	+12.5	+10.7	+10.8	+10.5	+12.7
FFT	47.3	SLC	+11.9	+11.6	+11.3	+11.4	+11.2	+11.5	+11.0	+11.0
		none	+11.7	+11.2	+11.3	+11.4	+11.3	+11.1	+11.2	+10.8
		buffer	+11.9	+11.9	+11.6	+11.8	+11.4	+11.4	+11.0	+10.8
FMM	52.4	SLC	+0.4	+0.4	+0.4	+0.4	+0.4	+0.5	+0.4	+0.4
		none	+0.4	+0.4	+0.5	+0.4	+0.4	+0.4	+0.4	+0.4
		buffer	+0.4	+0.3	+0.4	+0.4	+0.5	+0.4	+0.4	+0.4
GE	21.6	SLC	+30.5	+30.7	+31.4	+31.2	+31.7	+31.6	+31.4	+31.6
		none	+30.3	+30.5	+31.4	+31.0	+31.3	+31.3	+31.0	+31.0
		buffer	+30.7	+30.9	+31.8	+31.3	+31.8	+31.8	+31.5	+31.7
Ocean-Contig	49.7	SLC	-1.3	-2.8	-6.1	-3.5	-1.4	-3.6	-0.4	-3.6
		none	+3.2	+0.5	-1.0	-2.3	0.0	-2.6	-0.1	-1.1
		buffer	-2.4	+1.5	-1.5	-6.8	-0.2	+1.9	+0.8	-0.7
Ocean-Non-Contig	48.2	SLC	+7.8	+7.6	-6.3	+2.0	+4.1	+6.6	-8.3	-1.5
		none	+0.5	-3.6	+4.4	-11.3	+3.7	+4.7	+7.4	+3.3
		buffer	+4.5	+6.5	+5.8	+2.3	-0.2	+3.0	+3.7	+6.8
Water-Nsq	55.3	SLC	+0.2	+0.2	+0.2	+0.2	+0.2	+0.2	+0.2	+0.2
		none	+0.2	+0.2	+0.2	+0.2	+0.2	+0.2	+0.1	+0.2
		buffer	+0.2	+0.2	+0.2	+0.2	+0.2	+0.2	+0.2	+0.2

References

1. Craig Anderson and Anna R. Karlin. Two adaptive hybrid cache coherency protocols. In *the 2nd HPCA, San Jose, California*, pages 303–313, February 1996.
2. Satish Chandra et al. Where is time spent in message-passing and shared-memory programs? *6th ASPLOS, in SIGPLAN Notices*, 29(11):61–73, October 1994.
3. Chris Holt et al. The effects of latency, occupancy and bandwidth in distributed shared memory multiprocessors. Technical Report CSL-TR-95-660, Computer Systems Laboratory, Stanford University, January 1995.
4. Jeffrey Kuskin. *The FLASH Multiprocessor: designing a flexible and scalable system.* PhD thesis, Computer Systems Laboratory, Stanford University, November 1997. Also available as technical report CSL-TR-97-744.
5. Maged Michael and Ashwini Nanda. Design and performance of directory caches for scalable shared memory multiprocessors. In *the 5th HPCA, Orlando*, pages 142–151, January 1999.
6. Sarah A. M. Talbot. *Shared-Memory Multiprocessors with Stable Performance.* PhD thesis, Department of Computing, Imperial College, London, June 1999. Available on-line from `http://www.doc.ic.ac.uk/~samt/pub.html`.
7. Sarah A. M. Talbot and Paul H. J. Kelly. Reactive proxies: a flexible protocol extension to reduce ccNUMA node controller contention. In *Euro-Par 98*, volume 1470 of *LNCS*, pages 1062–1075. Springer-Verlag, September 1998.
8. B. A. Tanyi. *Iterative Solution of the Incompressible Navier-Stokes Equations on a Distributed Memory Parallel Computer.* PhD thesis, UMIST, 1993.
9. Manu Thapar and Bruce Delagi. Stanford distributed-directory protocol. *IEEE Computer*, 23(6):78–80, June 1990.
10. Steven Cameron Woo et al. The SPLASH-2 programs: characterization and methodological considerations. *22nd ISCA, in Computer Architecture News*, 23(2):24–36, 1995.