Eliminating Invalidation in Coherent-Cache Parallel Graph Reduction

Andrew J. Bennett and Paul H. J. Kelly

Department of Computing
Imperial College of Science, Technology and Medicine
London SW7 2BZ

Abstract. Parallel functional programs based on the graph reduction execution model display considerable locality of reference, favouring the use of large cache lines in the implementation of the shared heap on a shared-memory multiprocessor. They also display a very high rate of synchronisation, making conventional weakly-consistent coherency protocols ineffective at avoiding unnecessary contention for write access to cache lines due to false sharing. We present the design of a specially adapted cache coherency protocol and show results of simulation experiments which demonstrate that the protocol allows spatial locality to be exploited to at least the level of a conventional invalidation protocol, but without the unnecessary serialisation and network transactions caused by false sharing.

1 Introduction

Parallel graph reduction (PGR) uses a shared graph structure to manage and synchronise the parallel execution of a functional program. It provides a simple model of communication and synchronisation between processors with distinctive properties of importance in the design and optimisation of multicache implementations of shared-memory. The performance of parallel programs operating under this regime depends critically on the provision of access to the shared heap with very high average performance.

The most successful implementations of PGR (e.g. Goldberg's Buckwheat system [8] and Augustsson and Johnsson's $\langle\nu, G\rangle$-machine [3]) have used general-purpose shared-memory multiprocessors based on snooping cache coherency protocols [2]. For modest numbers of processors these systems implement a shared heap with near-ideal performance, but their size is limited by contention for the snooping bus.

In an earlier paper [5] we presented the results of a simulation study of parallel functional program execution on shared-memory multiprocessors using a simplified version of the standard invalidation-based multicache coherency protocol used on most modern multiprocessors. Such protocols provide a strong consistency property for application programs which make unrestricted use of reads, writes and synchronisations.

The results of that work can be summarised as follows:

- Parallel functional programs perform synchronisations at a far greater rate than more widely-studied program types (e.g. the SPLASH suite [14]).
- Many parallel functional programs display enough spatial locality to justify the use of very large cache lines, especially on architectures with high-bandwidth, high-latency interconnect as many current designs have.
- With large cache lines, the benefit of locality is annulled by the dramatic increase in communication caused by unnecessary invalidations when one cell on a cache line is updated, while the other cells remain stable.

In this paper we describe a modified cache coherency protocol designed specifically to support the parallel graph reduction model, and we show the results of simulated execution of a suite of benchmark programs to evaluate its success. We are especially concerned with the following questions:

- How effective is the new protocol at exploiting spatial locality and avoiding line stealing?
- How does performance of the PGR system operating under the new protocol compare with performance under the invalidation protocol?
- How does performance vary with interconnection network and program behaviour?

2 Exploiting the Memory Reference Characteristics of PGR

Objects in the shared-memory region are allocated, read, updated and shared in PGR quite differently from the way memory is used in typical imperative programs. For example, the following are inherent in the PGR model:

- There is a high turnover of cells, i.e. many cells are not accessed again soon after being created.
- Cells are updated at most once (i.e. to normal form). However, an implementation of PGR also requires a cell to be updated when a thread has gained the right to evaluate it.
- All writes to shared objects occur in critical sections (see Sect. 3).

The use of caching in the implementation of a shared-memory multiprocessor raises the question of how to prevent out-of-date cached data from continuing to be used indefinitely. Lamport defined sequential consistency [11] in which it is guaranteed that a read by any processor of any location will return the last value written to that location. Due to the unnecessary invalidation and serialisation incurred by this, protocols which offer weaker consistency properties have been proposed.

These advanced protocols are based on the concept of weak ordering [1], and coherency protocols based on weak ordering offer performance improvements by allowing invalidations generated by a thread whilst inside a critical section to be processed in parallel with computation. This increases concurrency by allowing

the processor to continue executing the instructions following the write without waiting for the invalidation to complete. The number of unnecessary invalidations caused by false sharing is therefore reduced, and coherence is enforced at synchronisation points only. Although a number of simulation studies of imperative programs have shown that significant improvements in performance can be achieved by using delayed consistency protocols (see for example [15]), the potential improvement is minor in this case due to the high rate of synchronisation inherent in PGR.

Our previous results indicate that it is line stealing (i.e. the unnecessary invalidation of lines caused by false sharing and subsequent line copying) which is the major cause of unnecessary network transactions when large cache lines are used with an invalidation protocol. If an update protocol had been used instead, these would have been eliminated, but active sharing of cache lines would be so great that a large proportion of writes would require update transactions. In summary, both invalidation and update protocols have significant problems. Instead we focus on avoiding the need for invalidations altogether.

3 A New Protocol

Objects in the shared-memory region are allocated, read, updated and shared in PGR quite differently from the way memory is used in typical imperative programs. For example most objects are created at runtime. Of particular importance is the three stage lifetime of nodes:

INACTIVE: The node has not been evaluated, and no thread has yet gained the right to evaluate it.

ACTIVE: The node has not been evaluated, but a thread has gained the right to evaluate it, and the node will be updated by its result by that thread at some time in the future.

EVALUATED: The node has been evaluated and will not be updated again.

The state transitions from INACTIVE to ACTIVE, and from ACTIVE to EVALUATED both require mutual exclusion: for a conventional shared-memory implementation, the state and lock fields of a node are used to achieve this. Each state transition requires write accesses to both the lock and the state field, and will result in any other copies of the cache line being invalidated. In the new protocol, ownership of cache lines is static, i.e. the processor that allocates nodes on a line remains the owner of that line. Any access made by a PE to a line it owns can be served by the local cache directly. Accesses to remote lines require network transactions only if a state transition is required. Copies of remote cache lines are made whenever a network transaction is made. Note that nodes in the EVALUATED state can be accessed from cached copies of lines directly. This is the only way in which locality of access can be exploited — locality of writes to remote cache lines cannot be exploited.

So, incoherent copies of cache lines can exist in the system since any access which may require a state transition to such a copy will result in a network

transaction with the owner of the line whose copy is always fully coherent. Evaluated nodes can be read from copies of lines since we can guarantee that they will not be updated.

False sharing still occurs, but line stealing is entirely eliminated since invalidation is not used. Note that a static ownership scheme greatly simplifies that task of locating the owner of a line: it is implicit in the address.

4 Experimental Design

The performance and behaviour of the new protocol have been assessed using a series of simulation experiments. A comprehensive description of the experimental design can be found in [4]. Here the most important aspects are discussed.

We have chosen to use execution-driven simulation which eliminates the validity problems of trace-driven simulation since the simulated processors read the data as it is at the simulated time at which the reference is made. A simplified architectural model is used, representing a 32-bit load-store architecture in which accesses to private memory take unit time and cache associativity and capacity effects are ignored. Although each assumption is invalid on a real machine, they prevent the results from being obscured by other effects.

The simulator models the state and copy set of each cache line in the shared-memory in order to determine the latency of each heap reference. Further information is also associated with each cell and cache line to enable the performance of the cache system to be closely monitored.

4.1 Source Language and Compiler

The source language is a lazy, higher-order functional language in the tradition of SASL, Miranda and Haskell [10]. The primary objective in building an optimising compiler for a lazy functional language is to reduce the frequency at which claims and references are made to the heap. It is therefore of great importance that the compiler used in our experiments should perform well. We have adopted the compiler developed for the FAST project [7], and although comparing compilers is difficult, we have some confidence that the system is competitive with the state of the art [9]. It also, conveniently, generates C, making generated code very easy to instrument and modify.

4.2 Garbage Collection

When storage allocated from the shared heap becomes free, it should be recycled for reuse. In a parallel system a parallel garbage collector is needed, and the area is the subject of intensive research. The behaviour of the garbage collector may interfere with normal program execution in two ways: firstly, it may change the relative timing of processes, depending on when it is activated, and whether all processors collect in synchrony. Secondly, garbage collection may substantially change the pattern in which store is allocated.

We have made an important simplification here: we have no garbage collection at all. Instead, each processor is allocated a large contiguous segment of the shared address space, and it allocates space from it starting from the base. This assumption illuminates one of our major objectives which is to learn general lessons about a large class of systems. We are less concerned that the experiments predict the actual performance of some production system. Although our system does not use garbage collection, the issue of how to collect in the presence of the new protocol must be addressed; we return to this in Sect. 6.

4.3 Example Parallel Functional Programs

The suite of programs we have been using are as follows:

pfib	compute the nth Fibonacci number
nqueens	compute a safe arrangement of queens on an $n \times n$ chess board
matmult	multiply two $n \times n$ matrices
quad	find the integral of a cubic function using adaptive quadrature
wave	tidal simulation of an estuary.

Wave divides the estuary into a matrix of sub-areas and the action of tides is simulated for some number of iterations. Each iteration produces three result matrices (represented by lists of lists) which are consumed by the following iteration.

Less trivial example programs are available, but the programs in this suite are simple enough to offer the possibility that their behaviour might be understood, while covering a variety of parallel program structures. The first four programs were used in Goldberg's Alfalfa and Buckwheat experiments, and are fully described in his thesis [8]. The last is taken from [16].

For space resaons in this paper we only present results from 3 programs: quad, matmult and wave.

4.4 Simulation of the New Protocol

Simulations were made of each program operating under the invalidatio and the new protocol using the following simulation parameters: 1, 2, 4, 8, 16, 32, 64 and 128 PEs, and cache line sizes of 1, 2, 4, 8, 16, 32, 64, 128, 256 cells. Note that, in order to make results easier to interpret, transaction latencies have been kept constant despite varying the cache line size. The memory timing model represents a modern multiprocessor in which the latency of a shared-memory access which requires use of the network is 500 times that of an access which can be satisfied by the local cache.

5 Simulation Results

The simulated execution time of a program is determined to a large extent by the latency of network transactions. Consequently, relative performance (i.e. speedup)

figures can be confusing. Since we want to compare the performance of two cache coherency protocols in a way that is independent of network latency, we have adopted the metric of "cache transaction ratio", representing the proportion of shared-memory accesses made by a program which require use of the network. Multicache schemes are designed to minimise average memory reference latency by minimising the cache transaction ratio.

5.1 Assessing Network Usage

Each graph in Fig. 1 shows the transaction ratio plotted against cache line size with separate curves representing different numbers of processors, the left column graphs were produced from simulations of the invalidation protocol, the right column from the new protocol.

The graph for quad with the new protocol shows that the transaction ratio decreases slightly as the line size is increased, regardless of the number of processors used. In the case of matmult and wave, the transaction ratio is initially very high (greater than 17% for 128PEs for matmult) but it falls rapidly for greater line sizes. These graphs indicate that using a large cache line size allows spatial locality to be exploited, resulting in a reduction in the load on the network and therefore execution time.

A comparison of these graphs with those produced for the invalidation protocol (left column of the figure) demonstrates the effectiveness of the new scheme: for each program the transaction ratios for the minimum line size are almost exactly the same, regardless of the number of PEs used. Although the reduction in the transaction ratio under the invalidation protocol as the line size is increased is initially as significant as under the new protocol, false sharing acts to reduce the improvement. For the new protocol, the transaction ratio continues to fall as the cache line size is increased, whereas the effect of false sharing acts to reduce the advantage of using large lines with the invalidation protocol. In the case of wave for example, the transaction ratio reaches a minimum at a line size of approximately 8 cells, after which it increases rapidly.

In summary:

- With the exception of wave, the transaction ratios observed for the new protocol at the minimum line size are very similar to those observed for the invalidation protocol regardless of the number of processors used. For wave they are approximately 20% greater than the corresponding figures for invalidation, regardless of the number of processors used.
- Increasing the line size usually produces a reduction in the transaction ratio, but never an increase. This indicates that the line stealing problem has been eliminated.
- For matmult and wave which both exhibited significant reductions in transaction ratio for the invalidation protocol as the line size was increased up to about 16 cells, the corresponding reduction with the new protocol is more significant.

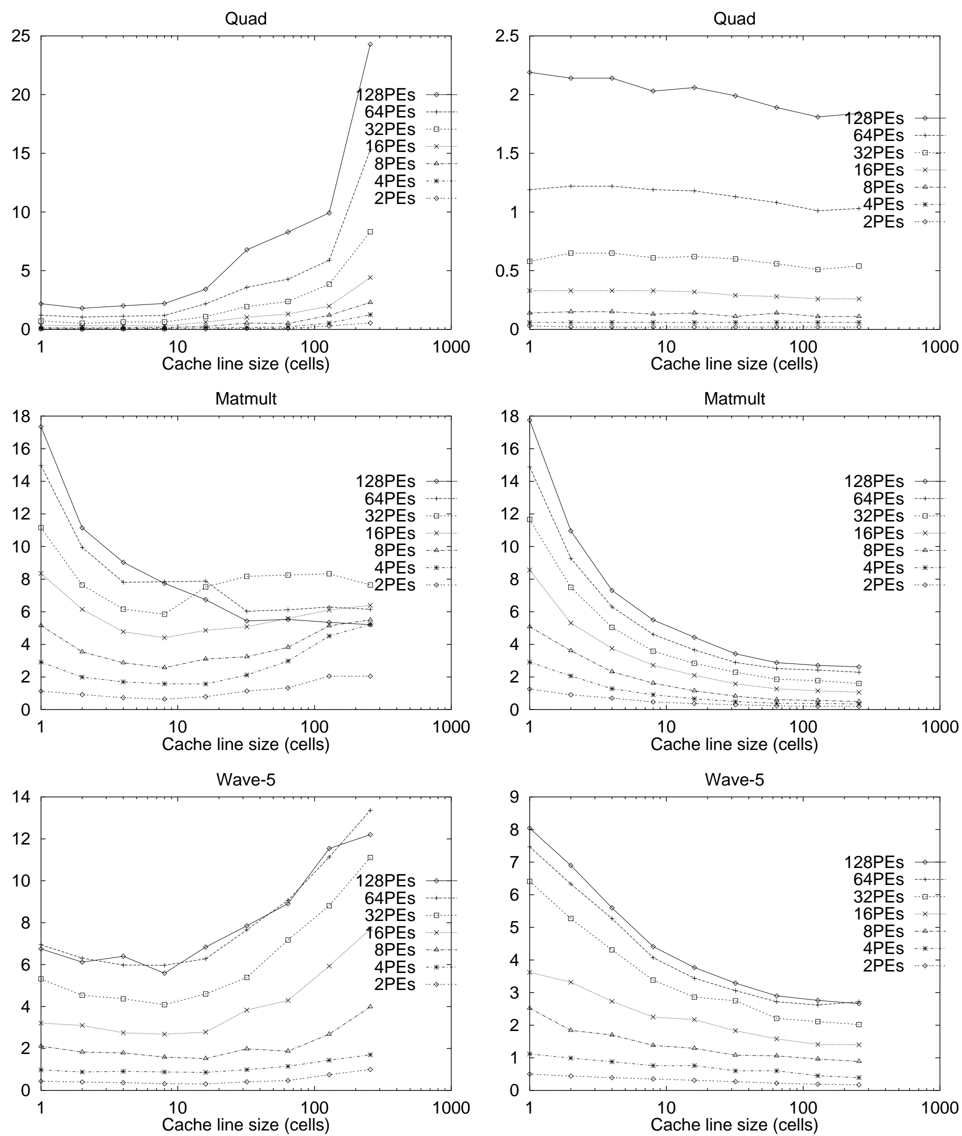

Fig. 1. Cache transaction ratio as a function of cache line size for quad, matmult and wave operating with the invalidation protocol (left column) and the new protocol (right column)

– Quad exhibited little reduction in transaction ratio as the line size was increased with the new protocol. This program does not use constructed data and therefore little spatial locality is present. Line stealing still exists with the invalidation protocol, leading to a very small optimum line size.

The results indicate that we have succeeded in exploiting spatial locality to at least the same extent as under the invalidation protocol and have also succeeded in eliminating unnecessary coherency transactions due to line stealing.

5.2 Exploiting Spatial Locality

In this section the advantage gained by exploiting spatial locality is assessed in detail.

The major difference between the invalidation protocol and the new protocol is the method used to acquire the right to evaluate and update remotely created cells. Using monitoring information this can be studied. Reducing cells which were created locally requires only low-latency memory references, but reducing remote cells requires high-latency references. Each call to the acquire procedure (which negotiates the right to evaluate cells) is classified according to the following scheme:

Simple : a call to acquire on a cell that was created locally, or a remotely created cell which has been accessed by this PE before (and is still in the cache).

Remote : a call to acquire on a cell which was created remotely and a network transaction is required (either the local cache did not have the cell or it was not a normal form).

Gain : the cell which was created remotely was present in normal form in the local cache and has not been accessed by the PE before. This is a benefit from spatial locality.

Counts of each type were made during simulations of each program with 32 processors. Graphs indicating the percentage of each type as a function of line size are shown in Fig. 2.

First consider quad: for the minimum line size, only 1.2% of acquires are to remote cells and incur the corresponding high latencies. This figure is only slightly affected by line size. Few gains are ever observed. This is to be expected since the only active sharing that occurs in the program is a direct result of cells being evaluated remotely, i.e. due to parallel evaluation.

The other programs are more interesting. Matmult shows that about 70% of acquires are to locally created cells, regardless of the size of cache lines. Most of the remaining 30% of acquires which need remote access for the minimum line size can be satisfied locally by using large lines. This is due to the data sharing behaviour of the program: the two input matrices are accessed by all threads, but since they are fully evaluated objects, large lines will allow spatial locality to be exploited. Some cells are created specifically to allow tasks to be spawned,

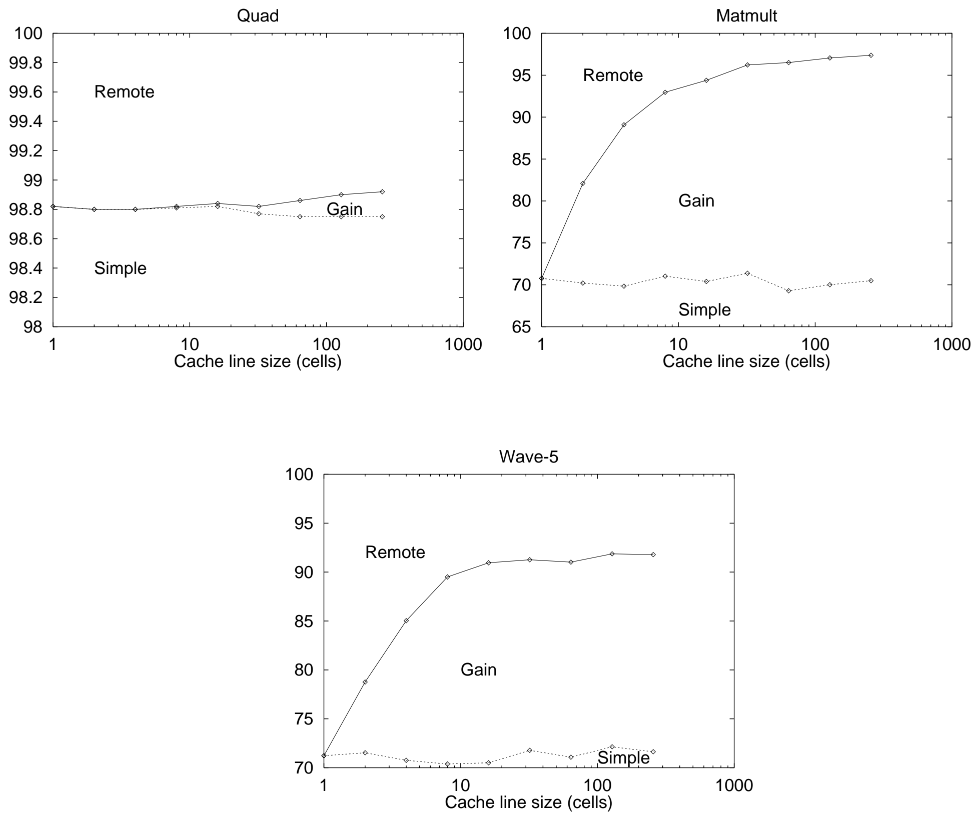

Fig. 2. Composition of acquire operations as a function of cache line size for 32 processor simulations of quad, matmult and wave operating with the new protocol

and therefore calling acquire on such cells will require network transactions. Consequently some acquires still require a network transaction, regardless of the line size used. Wave shows similar behaviour to matmult.

In summary:

- Gain acquires are negligible for programs which do not use constructed data; line size has little effect on the number of simple and remote acquires in this case.
- Gain acquires are significant for programs which do use constructed data, and spatial locality can be exploited by using large cache lines.

6 Discussion

The results shown above demonstrate that the new protocol allows spatial locality to be exploited to at least the level that was achieved with the invalidation protocol, and that line stealing caused by false sharing is eliminated.

An issue not addressed so far is the advantage offered by the new protocol being based on the static ownership of lines. A result of this is that the owner of a cache line is easy to determine — it is implicit in the address of the cell. This is in contrast to dynamic ownership schemes usually associated with invalidation protocols which require complicated and time consuming mechanisms to locate the owner of a line in the absence of a broadcast medium. Possible mechanisms include "probable ownership" chains in which each processor maintains a table recording the last known owner of each line. Locating the owner requires following the chain until the owner is found. Alternatively, each line can be assigned a manager through which ownership transfers must be negotiated. Both these schemes incur considerable overheads in terms of table space and extra messages, and are described more fully and evaluated experimentally in [13].

In order to simplify the presentation of results, the latencies of network transactions used above have been kept constant, despite varying the cache line size. Since the line stealing effect has been eliminated in this protocol, this results in the optimum line size being very large. It is important to note that other factors need to be considered when selecting a cache line size, and therefore a smaller line size would be used in practice.

The results do show, however, that the contention effect caused by false sharing does not need to be considered when selecting a line size for use with the new protocol.

Garbage collection in the presence of the new protocol raises a few problems, but also offers some advantages when compared to collection in the presence of a conventional invalidation protocol. We can view collection as a technique for reusing parts of the shared address space which have been used before. The new protocol presents the additional problem that cache lines can be freely cached and are not invalidated, and therefore a distributed garbage collector, such as a weighted reference count scheme [6, 17] cannot be used directly. However, a variant on a pure weighted count scheme [12] allows a mark-scan or copying collector to be used to collect local cells (which we have observed form the vast majority of cells), with weighted reference counting only being used for cells which have been accessed by more than one processor. In such a scheme, invalidation is still required, but there are two advantages:

- The region to be invalidated is large (a heap semi-space), thus reducing the volume of copy set data that must be maintained.
- The invalidation can be initiated immediately after a processor begins garbage collection, but need not be performed with respect to all processors in the copy set of the region until local collection has been completed. That is, the invalidation can take place at the same time as local collection.

7 Conclusions

In this paper, a new multicache coherency protocol has been proposed and evaluated for a set of benchmark functional programs. Using an invalidation protocol with delayed consistency (i.e. one which only enforces coherence at synchronisation points) offers little potential for performance improvements due to the high rate of synchronisation inherent in PGR. The new protocol is motivated by the memory access characteristics of PGR and allows spatial locality of read accesses to be exploited, but does not suffer from the line stealing problem. The ownership of cache lines is static, avoiding the need for complicated and costly mechanisms to locate the dynamic owner of lines and to record the copy set of each line.

Results of simulations indicate the following:

- Cache line size can have a significant effect on performance, but increasing line size will not reduce performance with the new protocol.
- Programs which did not use constructed data did not benefit from large lines, but did not suffer either. Programs which used constructed data can benefit from large lines due to spatial locality of read accesses.
- For each program the cache transaction ratio for the minimum line size is approximately equal to the corresponding figure for the invalidation protocol, but the reduction observed by increasing line size is more significant with the new protocol.
- The new protocol has the advantage that it uses static ownership of cache lines, simplifying the task of locating the owner of a line.
- The new protocol cannot take advantage of locality of write accesses to remotely created objects, but the results indicate that this did not lead to performance problems.

Although the negative effect of using large cache line sizes has been eliminated, the choice of line size is governed by a number of factors which must still be taken into account.

Future work will include a wider range of example programs, and the effect of garbage collection. Other ideas include the study of other coherent shared abstract data types which can be used for general-purpose portable parallel programming in conventional languages.

References

1. Sarita V. Adve and Mark D. Hill. Weak ordering — a new definition and some implications. Technical Report 902, Computer Sciences Department, University of Wisconsin-Madison, 1989.
2. James Archibald and Jean-Loup Baer. Cache coherence protocols: evaluation using a multiprocessor simulation model. *ACM Transactions on Computer Systems*, 4(4):273–298, November 1986.

3. Lennart Augustsson and Thomas Johnsson. Parallel graph reduction with the $\langle \nu, G \rangle$-machine. In *Fourth International Conference on Functional Programming Languages and Computer Architecture, London, September*, pages 202–213, 1989.
4. Andrew J. Bennett. *Parallel graph reduction for shared-memory architectures.* PhD thesis, Department of Computing, Imperial College, London, 1993.
5. Andrew J. Bennett and Paul H. J. Kelly. Locality and false sharing in coherent-cache parallel graph reduction. In Arndt Bode, Mike Reeve, and Gottfried Wolf, editors, *PARLE 93 Parallel Architectures and Languages Europe, Munich, June 1993*, volume 694 of *Lecture Notes in Computer Science*, pages 329–340, Berlin, 1993. Springer-Verlag.
6. D. I. Bevan. Distributed garbage collection using reference counting. In J. W. de Bakker, A. J. Nijman, and P. C. Treleaven, editors, *PARLE 87 Parallel Architectures and Languages Europe, Eindhoven, June 1987*, volume 259 of *Lecture Notes in Computer Science*, pages 176–187, Berlin, 1987. Springer-Verlag.
7. Stuart Cox, Shell-Ying Huang, Paul Kelly, Junxian Liu, and Frank Taylor. An implementation of static process networks. In D. Etiemble and J.-C Syre, editors, *PARLE 92 Parallel Architectures and Languages Europe, Paris, June 1992*, volume 605 of *Lecture Notes in Computer Science*, pages 497–512, Berlin, 1992. Springer-Verlag.
8. Benjamin F. Goldberg. *Multiprocessor Execution of Functional Programs.* PhD thesis, Yale University, New Haven, 1988.
9. Pieter H. Hartel and Koen G. Langendoen. Benchmarking implementations of lazy functional languages. In *Proceedings of the Conference on Functional Programming Langauges and Computer Architecture, Copenhagen, June*, 1993.
10. Paul Hudak, Simon L. Peyton Jones, and Philip Wadler. Report on the programming language Haskell — a non-strict purely functional language, version 1.2. *SIGPLAN Notices*, 27(5):1–162, May 1992.
11. Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, C-28(9):690–691, September 1979.
12. David R. Lester. An efficient distributed garbage collection algorithm. In E. Odijk, M. Rem, and J.-C Syre, editors, *PARLE 89 Parallel Architectures and Languages Europe, Eindhoven, June 1989*, volume 365 of *Lecture Notes in Computer Science*, pages 207–223, Berlin, 1989. Springer-Verlag.
13. Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4):321–359, November 1989.
14. Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta. SPLASH: Stanford Parallel Applications for Shared Memory. Technical Report CSL-TR-91-469, Computer Systems Laboratory, Stanford University, 1991.
15. Josep Torrellas and John Hennessy. Estimating the performance advantages of relaxing consistency in a shared-memory multiprocessor. In *International Conference on Parallel Processing, Pennsylvania State University, August*, pages 26–34, 1990.
16. Willem Vree. *Design Considerations for a Parallel Reduction Machine.* PhD thesis, University of Amsterdam, 1989.
17. Paul Watson and Ian Watson. An efficient garbage collection scheme for parallel computer architectures. In J. W. de Bakker, A. J. Nijman, and P. C. Treleaven, editors, *PARLE 87 Parallel Architectures and Languages Europe, Eindhoven, June 1987*, volume 259 of *Lecture Notes in Computer Science*, pages 432–443, Berlin, 1987. Springer-Verlag.