

Performance Models for Cluster-Enabled OpenMP Implementations

Jie Cai, Alistair P. Rendell, Peter E. Strazdins, H'sien Jin Wong
Department of Computer Science
The Australian National University
{Jie.Cai, Alistair.Rendell, Peter.Strazdins, Jin.Wong}@anu.edu.au

Abstract

A key issue for Cluster-enabled OpenMP implementations based on software Distributed Shared Memory (sDSM) systems, is maintaining the consistency of the shared memory space. This forms the major source of overhead for these systems, and is driven by the detection and servicing of page faults. This paper investigates how application performance can be modelled based on the number of page faults. Two simple models are proposed, one based on the number of page faults along the critical path of the computation, and one based on the aggregated numbers of page faults. Two different sDSM systems are considered. The models are evaluated using the OpenMP NAS Parallel Benchmarks on an 8-node AMD-based Gigabit Ethernet cluster. Both models gave estimates accurate to within 10% in most cases, with the critical path model showing slightly better accuracy; accuracy is lost if the underlying page faults cannot be overlapped, or if the application makes extensive use of the OpenMP flush directive.

1. Introduction

Cluster-enabled OpenMP implementations provide support for the shared memory OpenMP programming model on distributed memory cluster systems [2]. This offers not only increased portability for existing OpenMP programs, but also an alternative programming paradigm that is generally considered to be easier compared to the message passing model normally used on clusters. With the commercial release of the Intel Cluster OpenMP (CLOMP) compiler in 2006, interest in cluster-enabled OpenMP implementations is only likely to increase.

Most cluster-enabled OpenMP implementations are implemented over paged-based software Distributed Shared Memory (sDSM) systems [2, 12, 9]. This is viable since the relaxed memory consistency model of OpenMP limits the need to move globally shared memory pages between nodes to well defined consistency points (i.e. OpenMP barrier and

flush operations). Accesses to these shared memory pages are detected using page protection, with protection states of “Invalid”, “Read-Only”, and “Read-Write”. In general an access to an invalid page will cause the sDSM system to fetch the page from some other location in the cluster, while a write to a read-only page will cause the sDSM to duplicate the relevant page before allowing the write to proceed. The reason for the latter is so that multiple processes can write to distinct parts of the same memory page (by later using the copy to construct a “diff” that identifies where modifications have been made by that thread).

Servicing the various page faults and moving associated data between processes can be thought of as the overhead associated with maintaining memory consistency in the sDSM system. Taking a page fault interrupt or moving data the size of a memory page between nodes on a cluster is not, however, a trivial operation. Thus it is not surprising that these memory consistency operations are a major source of overhead in sDSM systems.

This paper explores two simple models that can be used to rationalize the performance of OpenMP applications running on clusters. In essence, these models propose that the performance of an OpenMP application running on a cluster can be estimated from the number and type of page faults encountered, and knowledge of the approximate cost of these events as obtained from running a simple OpenMP program on just two nodes of the cluster. To test the validity of our models we have used the OpenMP NAS Parallel Benchmarks (NPB) running on an 8 node Gigabit connected AMD cluster. Two fundamentally different cluster-enabled OpenMP implementations are considered, CLOMP [2] and Omni/SCLIB [12].

Related work is limited and not in the context of OpenMP. [7] presented a performance model based on fine-grained sDSM sub-operations and network parameters. Unlike ours, its purpose was to predict the performance changes in an sDSM over different hardware platforms. [8] developed a model for an object-based sDSM system, which had no element corresponding to the “diffing” overhead of page-based sDSM systems.

The rest of this paper is organized in six sections. In section 2 the background of the two sDSM systems is described. In section 3 the two different performance models are outlined. Section 4 contains details of the small program used to parameterize these models, while results for the NPB are given in section 5. Finally, section 6 contains our conclusions and comments for further work.

2. The sDSM Systems Modelled

The CLOMP and Omni/SCLIB OpenMP implementations for clusters are modelled in this paper. CLOMP is derived from the TreadMarks sDSM system [2], and uses a lazy release memory consistency model (LRC) [5]. It augments OpenMP to include a new `sharable` directive that is used to identify variables that can be referenced by more than one OpenMP thread. These variables are placed in globally addressable memory, that is created, maintained, and synchronized across all OpenMP threads through the sDSM system. The second cluster-enabled OpenMP implementation, Omni/SCLIB, uses the Omni OpenMP Compiler to transform an OpenMP program (C or Fortran) into intermediate C source code, that then makes use of the SCLIB sDSM system [12]. The approach used is similar to the earlier Omni/SCASH system of Sato et al [9].

Both the TreadMarks and SCLIB sDSM systems are *page-based*, partitioning the globally addressable memory into pages. In addition to this, pages in SCLIB are assigned a *home* node. The status of the page at its home node is what defines the most “up-to-date” view of that page. In contrast, the TreadMarks sDSM system [2] does not use page homes. Instead, changes made to a page by other nodes are patched into the local view when required. Naturally, “home-based” and “homeless” sDSM systems have different approaches for maintaining memory consistency. These approaches are detailed in the rest of this section.

2.1. CLOMP

In CLOMP, when the OMP code reaches a sequence point records are made of what pages have been modified since the last sequence point, and this information communicated to other threads as “write notices”. The aggregation of these write notices allows a thread to create a map detailing the pages that have remote modifications, and where those changes are located. As LRC is used, the actual page modifications are only obtained if and when they are required; an operation that may require the requesting thread to retrieve modifications from multiple other threads.¹

¹We note that CLOMP allows multiple threads to run within a single process on one node of a cluster. In which case there is some cooperation between all threads located within the same process, e.g. write notices are sent by just one thread on behalf of all the threads in that process.

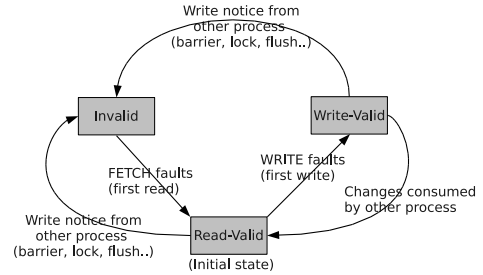


Figure 1: Major State Transfer Diagram for CLOMP (derived from [2], and experimental observation)

Transitions between the possible page states are given in Figure 1. Write notices are passed between threads when an OpenMP barrier, lock or flush directive is encountered. When a write notice is received for a shared page it will be set to “Invalid” so that a subsequent read or write request to that page will give rise to *fetch* faults that require *diffs* to be collected from the relevant other threads before the page can be used. Threads from which “diffs” have been requested (consumed) must change the protection for that page from “write-valid” to “read-valid” if necessary, indicating the start of a new “diff” reference point. Transitions from “read-valid” to “write-valid” occur the first time a write is made to a page, at which point a *write* fault will be issued necessitating the creation of a copy or “twin”.

2.2. Omni/SCLIB

Like the underlying TreadMarks sDSM that is used with CLOMP, SCLIB is a page-based sDSM system. Unlike TreadMarks, however, each page in the shared address space is assigned a *home* where the *master copy* of the page is maintained. This implies a fundamentally different approach to memory consistency. In particular, barrier operations involve all threads sending all their *diffs* to the relevant home threads so that the master copies can be brought up-to-date and invalidation notices sent back to all nodes holding copies of those pages. An access to an invalid page that occurs after a barrier is resolved by copying the up-to-date page back across the network from its home location.

Page state transitions depend on whether a page is local or remote. In the case of *local pages*, the transitions are between “Local Read-only” (LRO) and “Local Read-write” (LRW). LRO→LRW occurs upon a local page fault – referred to as a “local write fault”. Although the local page is the master copy and is, by definition, always the most up-to-date copy, detection of writes at the local node is necessary for the invalidation protocol used in barrier operations. The inverse, LRW→LRO, happens during the barrier in preparation for write detection during the next computation phase

(which lasts till the next barrier). Intuitively, local pages are set to LRO at initialization.

The actions required for *remote pages* are substantially more involved. These pages are set to “Remote Unmapped” (RUM) at initialization. A page fault on a RUM page is referred to as a “remote fetch fault” as this involves fetching the up-to-date copy from its home location and a page transition to “Remote Read-only” (RRO) – abbreviated as a RUM→RRO transition. A “remote write fault” occurs on a subsequent fault on a RRO page leading to the RRO→RRW transition. A twin is created during the handling of this fault so that differences may be determined later.

During a barrier operation, changes made to remote pages are sent back to the home nodes so that the master copy can be updated. Three different transitions are possible for remote pages depending on the current state of the page (RRO or RRW), and the number of writers that have modified the page since the last barrier. If any of the other nodes have made a change to the page, then the transition RRO→RUM or RRW→RUM are made. A RRW→RRO transition is used in the special case where that node is the only writer to the page.

The flush operation is implemented as a handshake protocol involving sending of changes made (if any) back to the home node and the subsequent return of the most up-to-date view of that data. This only needs to be done for remote pages as local pages are by definition always up-to-date. RRO pages will not have any changes and so will simply get updated, leaving the only page transition possible from flush operations as a RRW→RRO transitions occurring on the thread issuing the flush.

3. Performance Estimation Models

In this section two sDSM performance models are described. The first uses critical path analysis [3, 10] and requires a detailed knowledge of the number and type of page faults occurring for each thread in each parallel region. The second takes a more holistic approach requiring just the aggregate number of page faults occurring for all threads. Both the homeless and home-based sDSM systems will be considered, although the models will be developed initially within the context of the homeless sDSM system. We refer to both these models as SIGSEGV Driven Performance (SDP) models, reflecting the fact that the page faults give rise to SIGSEGV signals on POSIX compliant systems.

For both models, N^w and N^f will be used to denote the total number of *write* and *fetch* page faults respectively. The corresponding costs of these will be represented as C^w and C^f . For the home-based sDSM systems, a further distinction is made for the write faults depending on whether the fault happens for a local (N^{wl} and C^{wl}) or remote (N^{wr} and C^{wr}) page. It should be noted that the values for N are

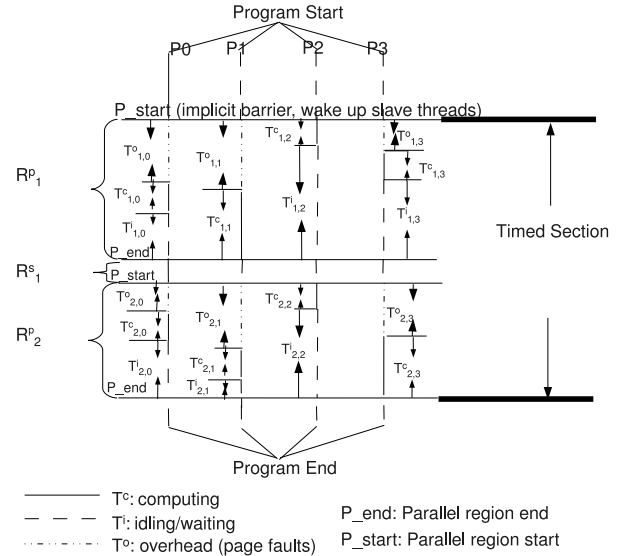


Figure 2: Schematic illustration of timing breakdown for parallel region using the SDP model

platform-independent, and if needs be these values may be gathered on a single node, provided the required number of sDSM processes are specified.

As per Amdahl’s law, the total execution time of an OpenMP program can be divided into serial and parallel regions. For the critical path SDP model the time spent in the parallel regions is further broken down into computation time (T^c , i.e. time spent doing useful work), overhead time (T^o , i.e. time spent servicing the various page faults), and idle time (T^i , i.e. time spent waiting for other threads). This is illustrated in Figure 2. The model assumes that page faults occurring on different threads can be fully overlapped, with a total per thread cost that can be estimated by knowing the cost of a page fault of a given type as measured when using just two nodes.

Both models assume that overheads, such as those associated with the creation of a parallel region, are small, at least in comparison to the memory consistency costs. Also, and although not central to the SDP models, we will make the assumptions that the OpenMP application is perfectly load-balanced (i.e. $T^i = 0$ and T^c is constant across threads) and that the total serial time is zero.

3.1 Homeless sDSM

If the total serial time is zero, the total elapsed time is simply $\sum_r T_r^{par}$, where T_r^{par} is the runtime for the r^{th} parallel region. Any given T_r^{par} is determined by the thread in parallel region r that has zero idle time. In the critical path model this is also the thread with the maximum value for

$T^c + T^o$. Denoting this time as T^{crit} , the value of T^{crit} for parallel region r is given by:

$$T_r^{crit} = \text{Max}_{i=0}^{p-1} (T_{r,i}^c + T_{r,i}^o) \quad (1)$$

where i is used to indicate thread id. (In Figure 2, T^{crit} occurs on $P1$ and $P3$ for parallel regions 1 and 2 respectively.)

The critical path model assumes that the values for T^o are determined solely by the number and type of page faults. For the homeless sDSM system these are either write or fetch page faults. Thus for parallel region r and thread i we have:

$$T_{r,i}^o = N_{r,i}^w C^w + N_{r,i}^f C^f \quad (2)$$

If we also assume that the code is perfectly parallelized, T^c in Equation (4) can be replaced by $T(1)_r^{par}/p$, where $T(1)_r^{par}$ is the elapsed time for parallel region r when the application is run using just one thread. Hence

$$T_r^{crit} = \frac{T(1)_r^{par}}{p} + \text{Max}_{i=0}^{p-1} (N_{r,i}^w C^w + N_{r,i}^f C^f) \quad (3)$$

and the total execution time on p processors becomes

$$Tot(p)^{crit} = \frac{Tot(1)}{p} + \sum_r \text{Max}_{i=0}^{p-1} (N_{r,i}^w C^w + N_{r,i}^f C^f) \quad (4)$$

In the above we have also used the fact that the total serial time is zero so $\sum_r T(1)_r^{par} = Tot(1)$.

Evaluating the time for the critical path SDP model requires a detailed knowledge of the number and types of page faults occurring in every parallel region. In our second, simplified SDP model, this is avoided. Instead only the aggregate SIGSEGVs count is required, and the total execution time is approximated as:

$$Tot(p)^{agg} = \frac{Tot(1)}{p} + (N^w C^w + N^f C^f) \left(f + \frac{1-f}{p} \right) \quad (5)$$

where f is a factor with a value between zero and one. The rationale behind the aggregate model is the idea that for any given application some fraction, f , of the total page faults will be serialised, while the remaining $(1-f)$ fraction will be able to be overlapped. What fraction this is will be application and environment dependent, and may vary as a function of the number of threads used. In this sense the model is empirical.

In comparison to the critical path approach we may expect:

$$Tot(p)_{f=0}^{agg} \leq Tot(p)^{crit} \leq Tot(p)_{f=1}^{agg} \quad (6)$$

Fully overlapped page faults ($f = 0$) will not occur if several threads request diffs from the same location at the same time. In which case we would expect the critical path approach to underestimate the total execution time and a large value for f to be necessary in the aggregate model.

For both models the cost of fetching a page, C^f , is based on a two-node measurement. For the homeless sDSM system, however, it is possible that diffs need to be fetched from many nodes. Recent work to design a memory consistency benchmark for OpenMP[11] suggests that if this occurs the cost will increase linearly with the number of nodes involved. This will cause the critical path model to underestimate the execution time as the number of threads increases. For the aggregate model it will require the value of f to increase as the number of threads increases in order to fit the observed data.

3.2 Home-based sDSM

For the SCLIB home-based sDSM system, Equations (4) and (5) are further broken down to include local and remote write faults. Additionally, there are two stages of data transferring: (1) fetch from the page home, when the reader/writer requests the updated page; (2) transfer back of the changes to the home after a write to a remote page and during the next barrier. The first operation is equivalent to a *fetch* fault in the homeless sDSM system, while the second operation is an extra cost, which we name “diffs-to-home”. The diffs-to-home is basically a some-to-one operation between the homes and the requesters. For SCLIB, the diffs are transferred one by one, using MPI non-blocking communications. When multiple nodes are transferring diffs back to home at the same time, each diff-to-home transfer will be treated as an individual transfer. We choose N^{dt} and C^{dt} to represent the number of diffs-to-home transfers and the associated cost of each. To make things easier, we make the assumption for SCLIB, that the size of diffs will be on average half of the largest possible size². Thus, C^{dt} can be considered as a constant on a certain hardware platform and for a certain problem size range. As only two nodes are involved in each diff transfer, the cost can be measured by running a normal ping-pong test.

Thus, the model for the home-based SCLIB sDSM system with the critical path and aggregate models are given by Equations (7) and (8) respectively, with the values also obeying Equation (6).

$$T^{crit} = \frac{Tot(1)}{p} + \sum_r \text{Max}_{i=0}^{p-1} (N_{r,i}^{wl} C^{wl} +$$

²For SCLIB, the page size is not fixed. The initial page size of SCLIB is 4096bytes and corresponding maximum diff size is 4681byte, which would increase by multiple of 4096 and 4681 bytes respectively with increasing problem size.

$$+ N_{r,i}^{wr} C^{wr} + N_{r,i}^f C^f + N^{dt} C^{dt}) \quad (7)$$

$$Tot(p)^{agg} = \frac{Tot(1)}{p} + (N^{wl} C^{wl} + N^{wr} C^{wr} +$$

$$+ N^f C^f + N^{dt} C^{dt}) (f + \frac{1-f}{p}) \quad (8)$$

With SCLIB a breakdown of both SDP models will occur for OpenMP applications that make heavy use of the flush directive. This is because SCLIB uses a simple flush-and-refresh protocol, where the differences are sent back to the home node, and then the most up-to-date version of the data returned. This operates at a region level, and does not involve page faults. For instance, if variable x is flushed, then only variable x is diffed and only variable x is refreshed, while the rest of the page where x resides is untouched. None of these operations are captured by page faults, so the SDP models will both underestimate the running time for flush intensive applications. For the aggregate model this effect can be masked by changing the value of the empirical fitting parameter f . Although, this could result in an unphysical value for f that is greater than one is needed in order to fit the available data.

4. Coefficient Measurement

The model detailed in section 3 consists of several coefficients that need to be measured. This is done using an OpenMP C program³ that involves various write and read operations on shared and private arrays. The first operation, WRITE(A) writes to the elements of A and returns the time it takes to complete that operation. Similarly, READ(A) will read the elements in A and return the time required.

Figure 3 show how the coefficients can be measured. The algorithm uses private (R) and shared (S) arrays of equal length. For the home-based sDSM, runtime options are used to ensure that the pages in the shared array S will have home locations on the same node as thread-0. The time it takes to perform the WRITE and READ operations on the private array R are recorded as D^w and D^r respectively (lines 1 and 2). This provides a reference time for performing the memory operations without shared memory consistency concerns.

The measurements for C^w , C^f , C^{wr} , and C^{wl} can now be done. Two threads are used in this code. In line 3 and 4, a READ(S) is done by both threads followed by a barrier operation. For CLOMP, this step ensures that the pages of S begin in a read-only state. Next thread-0 performs a write operation on S . For the homeless sDSM system the value recorded is C^w , while for the home-based sDSM system

1. $D^w \leftarrow \text{WRITE}(R)$
2. $D^r \leftarrow \text{READ}(R)$
3. READ(S)
4. barrier
5. **if** thread-0:
6. $C^{wl} \leftarrow (\text{WRITE}(S) - D^w)/\text{npages}$
7. barrier
8. **if** thread-1:
9. $C^f \leftarrow (\text{READ}(S) - D^r)/\text{npages}$
10. $C^{wr} \leftarrow (\text{WRITE}(S) - D^w)/\text{npages}$

Figure 3: The algorithm used to determine the SDP coefficients. The code shown is in a parallel region. R is a private array while S is a shared one. Variables D^w and D^r represent reference times for accessing private array R .

the time recorded is C^{wl} . At the end of the write phase a barrier operation is performed.

Lines 9 and 10 are then executed by thread-1 to give values for coefficients C^f and C^{wr} . Note also that, as with the previous write measurement, the value of C^{wr} on the home-based sDSM system is the same as C^w on the homeless sDSM system.

The number and type of page faults were obtained for SCLIB by modifying the code. For CLOMP we have used the SEGVprof profiling tool that is provided as part of the distribution. This tool creates profile files (.gmon files) for all CLOMP processes, reporting the segmentation faults occurring for each thread in each parallel region. SEGVprof provides a script (segvprof.pl) that reports aggregated results, and this was extended to produce per-thread results.

5. Experimental Results

To test the applicability of the SDP models, we ran the OpenMP version of the NAS Parallel Benchmark (NPB)[1, 4] (NPB-OMP) on an 8 nodes AMD cluster. Each cluster node had an Athlon dual core 2.2 GHz CPU with 4 GB of memory. The nodes were connected using Gigabit Ethernet. The OSU benchmark [6] gave a ping-pong latency and bandwidth for the cluster of 62.5us and 103.4MB/s respectively.

The code outlined in Section 4 was used to measure the timing coefficients for the SDP models. This gave values of $C^w=21.6\mu\text{s}$ and $C^f=320.1\mu\text{s}$ for the homeless sDSM, and $C^{wr}=40.5\mu\text{s}$, $C^{wl}=20.9\mu\text{s}$ and $C^f=295.3\mu\text{s}$ for the home-based sDSMs. Results for the homeless and home-based systems will now be considered in detail.

³The source code of the program is available at http://ccnuma.anu.edu.au/dsm/segv_cost.

5.1. Homeless sDSM Results

The EP, SP, BT, FT, LU, IS and CG class A and C benchmarks from the NPB-OMP suite were used. Both elapsed time and the associated fault counts have been recorded at a per thread level for the parallel regions within the timed section of each NPB benchmark. The number of read and write page faults along the critical paths are shown in Table 1. The observed sequential and parallel execution times together with those predicted from the critical path and aggregate SDP models are given in Table 2.

The most immediate observation from Table 1 is the lack of page faults for EP. This suggests this benchmark will scale equally well on CLOMP as for a non-sDSM OpenMP implementation. Differences between the number of write faults and fetch faults gives an estimate of the number of read only pages. In most cases this difference is relatively small, the exception being for the IS benchmark where fetches typically outnumber writes by a factor of two. In a few cases fetch faults are less than the number of writes, this suggests that there are local changes to globally shared pages that are not subsequently requested by other threads. For good scalability the page fault count along the critical path should decrease as the number of threads increases. This is true to some level for all benchmarks, except IS.

Whether a particular benchmark performs well using CLOMP will depend on how the total number of page faults compares with the overall execution time. Ignoring EP the page fault counts given in Table 1 are found to vary by around four orders of magnitude between the different benchmarks in a given class. On the other hand the sequential execution times given in Table 2 vary by about two orders of magnitude. This suggests that the different benchmarks will show very different behavior when run using CLOMP. This is indeed the case as evident from the observed times given in Table 2, where speedups for eight processors range from around eight for EP to a slowdown of two or more for CG class A or LU class C. For this reason the NPB OMP suite represents an interesting and challenging problem set for cluster-enabled OpenMP implementations.

5.1.1 Critical path model estimates

The estimated elapsed times obtained using the critical path model are shown in Table 2. With the exception of EP, and the class C IS and CG benchmarks the observed parallel performance is poor, suggesting that overhead dominates performance. While this implies that applications of this type and size should not be run in parallel on this cluster using CLOMP, the objective of the work presented here is different; namely to consider how well the proposed models predict the above scalability, whatever that is.

For class A the critical path model predicts speedup rea-

Table 1: Critical path page faults counts for the NPB-OMP benchmarks run using CLOMP

Bench- marks	2 threads		4 threads		8 threads	
	Write	Fetch	Write	Fetch	Write	Fetch
	Class A					
EP	0	0	0	0	0	0
SP	8.51E+5	8.38E+5	7.36E+5	7.82E+5	5.66E+5	6.39E+5
BT	4.91E+5	4.93E+5	4.07E+5	4.45E+5	2.77E+5	3.51E+5
FT	1.18E+5	1.18E+5	8.83E+4	8.96E+4	5.22E+4	5.32E+4
LU	1.12E+6	1.17E+6	8.59E+5	1.03E+6	5.90E+5	8.28E+5
IS	2.56E+3	5.10E+3	3.84E+3	7.66E+3	4.48E+3	8.93E+3
CG	8.54E+3	8.95E+3	6.21E+3	1.32E+4	4.67E+3	1.47E+4
	Class C					
EP	0	0	0	0	0	0
SP	1.17E+7	1.16E+7	9.57E+6	9.66E+6	8.05E+6	8.21E+6
BT	7.31E+6	7.31E+6	5.56E+6	5.94E+6	4.04E+6	4.70E+6
LU	1.58E+7	1.68E+7	9.68E+6	1.20E+7	5.57E+6	7.42E+6
IS	4.10E+4	8.19E+4	6.14E+4	1.23E+5	7.17E+4	1.43E+5
CG	3.02E+5	3.03E+5	1.60E+5	4.53E+5	8.82E+4	5.26E+5

sonably well, except for LU and CG. For these cases the predicted performances are better than the observed ones. The fact that the errors in the predicted values get larger with increasing thread count can be attributed to the onset of contention; something that is ignored in the critical path model. The relatively poor agreement for two threads indicates, however, that for these two benchmarks there are other issues beyond contention. For LU the problem appears to be significant load imbalance that is not accounted for by the current version of the critical path model. (When LU is run using two threads with a non-sDSM OpenMP implementation on one of the dual core nodes of the AMD cluster a speedup of just 1.3 is observed.) For CG the problem is that this benchmark contains significant serial regions (~ 1.4 sec out of 5.8sec) and this is again not included in the current version of the critical path model.

The results for the class C benchmarks show similar trends to those observed for class A. For CG the sequential time is, however, a much smaller fraction of the overall execution time, so the predicted values are now in slightly better agreement with the observed results. The results for IS are interesting, in that the critical path model noticeably underestimates the observed parallel performance. This appears to be due to cache effects that give smaller than predicted computational times on multiple processors (i.e. less than T^c/p). (Using performance counters the level 2 cache hit ratio for IS is $\sim 9\%$ when using one thread, but $\sim 26\%$ on the master thread when using eight threads.)

In summary, the relative errors for the class C NPB are smaller than for class A. For both classes the error in the critical path model increases with number of threads used, but this is expected given that the model assumes no contention. The largest deviations occur for LU and CG, with the other benchmarks giving errors that are typically 10% or less.

5.1.2 Aggregate model estimates

In contrast to the critical path model the aggregate model includes an empirical parameter f that can be adjusted to

Table 2: Comparison between observed and estimated speedup for running NPB class A and C on the AMD cluster with CLOMP

Bench- mark	Seq. Time (sec)	# of thds	S_{Obs}	S_{Crit}	$S_{agg}(f)$ for various f					
					0	0.25	0.5	0.75	1	
Class A										
EP	26.47	2	2.05	2.00	2.00	2.00	2.00	2.00	2.00	2.00
		4	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
		8	8.25	8.00	8.00	8.00	8.00	8.00	8.00	8.00
SP	137.4	2	0.41	0.39	0.39	0.32	0.28	0.24	0.21	0.20
		4	0.45	0.46	0.47	0.28	0.20	0.16	0.13	0.13
		8	0.50	0.59	0.61	0.23	0.15	0.11	0.08	0.08
BT	145.4	2	0.64	0.60	0.60	0.51	0.45	0.40	0.35	0.35
		4	0.77	0.77	0.80	0.50	0.36	0.28	0.23	0.23
		8	0.95	1.06	1.11	0.44	0.28	0.20	0.16	0.16
FT	12.0	2	0.27	0.26	0.26	0.21	0.18	0.16	0.14	0.14
		4	0.37	0.36	0.36	0.21	0.15	0.12	0.10	0.10
		8	0.59	0.61	0.62	0.24	0.15	0.11	0.08	0.08
LU	197.2	2	0.29	0.40	0.40	0.33	0.28	0.25	0.22	0.22
		4	0.31	0.52	0.53	0.32	0.23	0.18	0.15	0.15
		8	0.31	0.65	0.70	0.27	0.17	0.12	0.10	0.10
IS	3.4	2	1.03	1.00	1.13	1.02	0.93	0.85	0.79	0.79
		4	1.10	1.00	1.36	0.91	0.68	0.55	0.46	0.46
		8	1.06	1.00	1.58	0.66	0.41	0.30	0.24	0.24
CG	5.8	2	0.78	0.97	0.98	0.87	0.78	0.71	0.65	0.65
		4	0.64	1.00	1.04	0.67	0.49	0.39	0.32	0.32
		8	0.39	1.05	1.07	0.43	0.27	0.19	0.15	0.15
Class C										
EP	428.5	2	2.04	2.00	2.00	2.00	2.00	2.00	2.00	2.00
		4	4.08	4.00	4.00	4.00	4.00	4.00	4.00	4.00
		8	8.19	8.00	8.00	8.00	8.00	8.00	8.00	8.00
SP	2346.9	2	0.48	0.46	0.46	0.38	0.33	0.29	0.26	0.26
		4	0.59	0.60	0.62	0.38	0.27	0.21	0.17	0.17
		8	0.71	0.76	0.87	0.34	0.21	0.15	0.12	0.12
BT	2767.3	2	0.75	0.71	0.71	0.61	0.54	0.48	0.43	0.43
		4	1.01	1.02	1.03	0.66	0.49	0.39	0.32	0.32
		8	1.35	1.42	1.59	0.66	0.42	0.30	0.24	0.24
LU	3284.6	2	0.33	0.44	0.44	0.37	0.32	0.28	0.25	0.25
		4	0.43	0.67	0.68	0.42	0.30	0.24	0.20	0.20
		8	0.60	1.13	1.14	0.46	0.29	0.21	0.16	0.16
IS	113.9	2	1.50	1.35	1.46	1.37	1.29	1.21	1.15	1.15
		4	1.87	1.64	2.08	1.53	1.21	1.00	0.85	0.85
		8	2.15	1.85	2.72	1.26	0.82	0.61	0.48	0.48
CG	1385.7	2	1.68	1.74	1.74	1.68	1.63	1.58	1.54	1.54
		4	2.71	2.80	2.80	2.29	1.93	1.67	1.48	1.48
		8	3.24	4.03	4.04	2.16	1.48	1.12	0.90	0.90

account for contention. Results for various values of f are shown in Table 2. As EP contains no page faults, the predicted results are independent of f . For all other benchmarks, except IS, the aggregate model with two threads and $f = 0$ agrees very well with the critical path model. This indicates that the numbers of both fetch and write faults occurring on each thread are roughly equal. For IS this is not true, rather there are roughly double the number of fetch faults on one thread as the other, and a similar but opposite imbalance for the write faults.

With increasing thread count the difference between $S_{agg}(0)$ and S_{crit} tends to increase, indicating greater imbalance between the numbers of each type of page faults on each thread. In all cases Equation (6) holds.

While differences between $S_{agg}(0)$ and S_{crit} reflect imbalance in the numbers of page faults of a given type occurring on the different threads, contention will cause both $S_{agg}(0)$ and S_{crit} to over estimate S_{obs} . Contention is expected to increase with increasing thread count, implying that the value of f required by the aggregate model in order to fit the observed data is likely to increase with thread count. This trend is apparent for SP, BT and FT, where we find $S_{agg}(f=0)$ to be very accurate for two threads, but a non zero value of f to be more applicable on eight threads (even accounting for the difference between S_{crit} and $S_{agg}(0)$). For LU, IS and CG a similar analysis is much harder given

Table 3: Comparison between observed and estimated speedup for running NPB class A and C on the AMD cluster with SCLIB

Bench- mark	Seq. Time (sec)	# of thds	S_{Obs}	S_{Crit}	$S_{agg}(f)$ for various f					
					0	0.25	0.5	0.75	1	
Class A										
EP	33.2	2	1.99	2.00	2.00	2.00	2.00	2.00	2.00	2.00
		4	3.99	4.00	4.00	4.00	4.00	4.00	4.00	4.00
		8	7.87	8.00	8.00	7.99	7.99	7.99	7.99	7.99
SP	218.1	2	0.15	0.14	0.14	0.11	0.09	0.08	0.07	0.07
		4	0.17	0.18	0.18	0.10	0.07	0.06	0.05	0.05
		8	0.26	0.28	0.29	0.11	0.07	0.05	0.04	0.04
BT	187.46	2	0.32	0.29	0.29	0.23	0.20	0.17	0.15	0.15
		4	0.36	0.37	0.37	0.22	0.16	0.12	0.10	0.10
		8	0.54	0.57	0.58	0.22	0.14	0.10	0.08	0.08
FT	20.1	2	0.16	0.25	0.25	0.21	0.18	0.15	0.13	0.13
		4	0.25	0.33	0.34	0.20	0.14	0.11	0.09	0.09
		8	0.41	0.56	0.56	0.21	0.13	0.10	0.08	0.08
LU	269.4	2	0.01	0.30	0.30	0.25	0.21	0.18	0.16	0.16
		4	0.01	0.37	0.37	0.22	0.16	0.12	0.10	0.10
		8	0.02	0.55	0.57	0.22	0.13	0.10	0.08	0.08
CG	5.5	2	0.22	0.18	0.18	0.15	0.13	0.11	0.10	0.10
		4	0.31	0.23	0.23	0.14	0.10	0.08	0.06	0.06
		8	0.38	0.32	0.34	0.13	0.08	0.06	0.04	0.04
Class C										
EP	509.1	2	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00
		4	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
		8	7.97	8.00	8.00	8.00	8.00	8.00	8.00	8.00
SP	2869.4	2	0.24	0.23	0.23	0.19	0.16	0.14	0.12	0.12
		4	0.22	0.31	0.31	0.18	0.13	0.10	0.08	0.08
		8	0.32	0.51	0.52	0.20	0.12	0.09	0.07	0.07
BT	3162.9	2	0.56	0.55	0.55	0.47	0.40	0.36	0.32	0.32
		4	0.60	0.78	0.79	0.49	0.36	0.28	0.23	0.23
		8	0.84	1.28	1.33	0.54	0.34	0.25	0.19	0.19
CG	1361.8	2	1.12	1.02	1.03	0.92	0.83	0.76	0.70	0.70
		4	1.88	1.59	1.63	1.12	0.86	0.70	0.58	0.58
		8	1.94	2.70	2.80	1.31	0.86	0.64	0.51	0.51

that the values for S_{obs} are contaminated by the load imbalance, serial regions, and cache issues discussed above. Overall, however, the results for all the NPB benchmarks given in Table 2 clearly show that the basic cost of servicing the various page faults primarily determines performance (or lack thereof), and contention issues are secondary.

5.2. Home-based sDSM Results

Currently the Omni compiler cannot compile IS, as it does not support the latest version of C header file. As a consequence this benchmark was not considered when using SCLIB. For the other benchmarks the page size used when running the class A benchmarks was 4096 bytes, except for FT, which used a page size of 8192 bytes. For the class C, SP, BT and LU benchmarks the page size was 16384 bytes, for CG it was 20480 bytes, and for EP it was 4096 bytes. The cost of diff-to-home was measured to be ~ 120 us for EP class C and all class A benchmarks except FT (190us). For BT/SP/LU class C the value was 272us, while for CG it was 312us. Using Equation (7) and (8) the predicted performances when using SCLIB on the AMD cluster are given in Table 3.

For the class A benchmarks predicted performance shows similar trends to those observed for the homeless sDSM; the critical path model agrees well with the $f = 0$ aggregate model, with both tracking the observed results except for LU and CG. For CG the disagreement is less pronounced compared to the homeless sDSM, but this in part reflects the fact that the slowdown with SCLIB is much greater, rendering the serial portion of this benchmark less

Table 4: Average relative errors for the predicted NPB speedups evaluated using the critical path and aggregate ($f = 0$) SDP models and data from Tables 2 and 3.

		Crit			agg($f=0$)		
		2 threads	4 threads	8 threads	2 threads	4 threads	8 threads
CLOMP	ClassA	0.12	0.20	0.46	0.13	0.24	0.57
	ClassC	0.10	0.13	0.24	0.09	0.14	0.31
Without LU & CG	Class A	0.04	0.03	0.08	0.05	0.07	0.19
	Class C	0.05	0.04	0.07	0.04	0.05	0.17
SCLIB ^a	ClassA	0.18	0.13	0.13	0.18	0.14	0.14
	ClassC	0.04	0.22	0.38	0.03	0.22	0.41

^a Without LU

significant. For LU the predicted performance is very different to the observed performance, with the observed performance being very poor. The reason for this is the same in both cases. Namely the LU benchmark makes extensive use of flush operations to synchronise threads. As the flush directive in SCLIB does not cause a page fault both the critical path and aggregate model will not account for this cost and will therefore considerably overestimate performance (note even the aggregate model with $f = 1$ is unable to account for the observed slow-down). As the flush operations keep the SCLIB event handler busy, this greatly slows down the program. This issue has been recognised, and alternative approaches to implementing flush in SCLIB are discussed elsewhere [12]. In the meantime we have omitted LU from the class C results given in Table 3.

For class C the effect of contention in the system is more noticeable, with the critical path and $f = 0$ aggregate model predictions for 4 and 8 threads being significantly overestimates. This is due to multiple threads requesting pages from the same home location at the same time. In this respect we note that the use of larger pages can to some extent increase the likelihood of threads sharing pages and possible associated contention.

6. Conclusion

In Table 4 we summarise the overall accuracy of the critical path and aggregate model ($f = 0$) for the CLOMP and SCLIB sDSM systems using a variety of different thread counts. These results show that for CLOMP the model based on critical path analysis is slightly more accurate than the simpler one based on aggregate page fault counts. For computations where the page fault overhead from different threads is highly overlapped, the models are generally accurate to within 10%, and when this is not the case, the models are optimistic. For the SCLIB sDSM the model performs particularly badly for benchmarks, like LU, that include significant number of flush operations. But even excluding LU, the accuracies obtained for SCLIB are noticeably worse than for CLOMP.

The accuracy of the current models are known to be limited for applications that contain significant serial regions

and/or load imbalance. Future work will investigate ways to include this in the SDP models. Also, although the critical path model is in general more accurate than the aggregate approach, it lacks any concept of contention. And while this can be included in the aggregate approach, it is retrospective in that f is adjusted to fit the observed data. To develop a more robust way for including contention further analysis of the behavior of the NPB and other applications on a variety of cluster platforms is required. Also developing profiling tools to assist in this endeavour would be useful.

7. Acknowledgments

This work is part funded by Australian Research Council Grant LP0669726 and Intel Corporation, and was supported by the APAC National Facility at the ANU.

References

- [1] D. H. B. et. al. The NAS Parallel Benchmarks. *Technical Report RNR-94-007, NASA Ames Research Center*, 1994.
- [2] J. P. Hoeflinger. Extending OpenMP to clusters. *White Paper, Intel Corporation*, 2006.
- [3] J. Hollingsworth. Critical path profiling of message passing and shared-memory programs. *Parallel and Distributed Systems, IEEE Transactions on*, 9(10):1029–1040, Oct 1998.
- [4] H. Jin, M. Frumkin, and J. Yan. The OpenMP implementation of NAS parallel benchmarks and its performance. *Technical Report: NAS-99-011*, 1999.
- [5] P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy release consistency for software distributed shared memory. In *Proc. of the 19th Annual Int'l Symp. on Computer Architecture (ISCA'92)*, pages 13–21, 1992.
- [6] J. Liu, B. Chandrasekaran, W. Yu, J. Wu, D. Buntinas, S. Kini, D. Panda, and P. Wyckoff. Microbenchmark performance comparison of high-speed cluster interconnects. *Micro, IEEE*, 24(1):42–51, Jan.-Feb. 2004.
- [7] E. W. Parsons, M. Brorsson, and K. C. Sevcik. Predicting the performance of distributed virtual shared-memory applications. *IBM Syst. J.*, 36(4):527–549, 1997.
- [8] L. Peng, W.-F. Wong, and C.-K. Yuen. The performance model of SilkRoad - a multithreaded DSM system for clusters. In *Proceedings of CCGRID '03*, page 495, 2003.
- [9] M. Sato, H. Harada, and Y. Ishikawa. OpenMP compiler for a software distributed shared memory system SCASH. In *WOMPAT2000*, July 2000.
- [10] M. Schulz. Extracting critical path graphs from MPI applications. *Cluster Computing, 2005. IEEE International*, pages 1–10, Sept. 2005.
- [11] H. J. Wong, J. Cai, A. P. Rendell, and P. E. Strazdins. Microbenchmarks for cluster OpenMP implementations: Memory consistency costs. In *IWOMP 2008, LNCS 5004*, pages 60–70, 2008.
- [12] H. J. Wong and A. P. Rendell. The design of MPI based distributed shared memory systems to support OpenMP on clusters. In *Cluster07, IEEE Catalog Number 07EX1855C*, pages 231–240, 2007.