Region-Based Prefetch Techniques for Software Distributed Shared Memory Systems

Jie Cai [#], Peter E. Strazdins [*], and Alistair P. Rendell [+]
School of Computer Science
The Australian National University
Canberra, ACT, Australia
[#] Jie.Cai@anu.edu.au
[*] Peter.Strazdins@anu.edu.au
[+] Alistair.Rendell@anu.edu.au

***Abstract*—Although shared memory programming models show good programmability compared to message passing programming models, their implementation by page-based software distributed shared memory systems usually suffers from high memory consistency costs. The major part of these costs is inter-node data transfer for keeping virtual shared memory consistent. A good prefetch strategy can reduce this cost. We develop two prefetch techniques, TReP and HReP, which are based on the execution history of each parallel region. These techniques are evaluated using offline simulations with the NAS Parallel Benchmarks and the LINPACK benchmark.**

On average, TReP achieves an efficiency (ratio of pages prefetched that were subsequently accessed) of 96% and a coverage (ratio of access faults avoided by prefetches) of 65%. HReP achieves an efficiency of 91% but has a coverage of 79%. Treating the cost of an incorrectly prefetched page to be equivalent to that of a miss, these techniques have an effective page miss rate of 63% and 71% respectively. Additionally, these two techniques are compared with two well-known software distributed shared memory (sDSM) prefetch techniques, Adaptive++ and TODFCM. TReP effectively reduces page miss rate by 53% and 34% more, and HReP effectively reduces page miss rate by 62% and 43% more, compared to Adaptive++ and TODFCM respectively. As for Adaptive++, these techniques also permit bulk prefetching for pages predicted using temporal locality, amortizing network communication costs and permitting bandwidth improvement from multi-rail network interfaces.

I. Introduction

Due to the high complexity of the message passing programming paradigm on cluster systems, there is much interest in developing alternative programming paradigms that are easier to use but without significant performance penalties. The most widely used such paradigm, the shared memory programming model, can be facilitated on clusters via software Distributed Shared Memory (sDSM) systems, which have the advantage of both the programmability of shared memory programming models and the low cost of distributed memory architectures. sDSM systems designed with OpenMP compiler support are also known as cluster-enabled OpenMP systems [1], [2], [3], [4], [5].

With the release of Intel Cluster OpenMP in 2006, cluster-enabled OpenMP systems became the mainstream of sDSM. Although cluster-enabled OpenMP systems show acceptable performance on some scientific applications, for example selected workloads from the Gaussian Quantum Chemistry code (Gaussian03) [6], they still suffer from high memory consistency costs [7], [8]. The major part of the cost is inter-node data transfer, due to the memory consistency work within global synchronization points [9]. An effective prefetch strategy can reduce memory consistency costs by overlapping computation and communication. Furthermore, if prefetched pages are aggregated, it becomes possible to amortise communication latencies and to leverage multirail techniques to achieve better inter-node bandwidth [9], [10].

However, it is quite complex to effectively prefetch in cluster-enabled OpenMP systems due to two major problems. The first is that the simple techniques proposed for hardware prefetches are not usually effective for sDSMs, because the larger granularity of access (page-level vs cache-line level) hinders prediction and the much higher latencies requires prefetches to be issued much earlier. The second reason is that prefetches generate significant overhead when they are issued unnecessarily or incorrectly [11].

In this paper, we have designed a set of Region-based Prefetch (ReP) techniques which address the first problem by considering how the patterns of page accesses vary both across executions and within the execution of each identifiable parallel and serial region. The second problem is addressed by avoiding the issue of prefetches for non-repeatable regions or non-repeatable page access patterns.

There are some existing prefetch techniques for sDSM systems, including Dynamic Aggregation technique [12] designed by Amza *et al.*, B+ [13] and Adaptive++ [11] designed by Bianchini *et al.*, and the third-order differential finite context method (TODFCM) [14] adopted by the Delphi sDSM system [15] by Speight *et al.*. This paper compares two new ReP techniques with some of these existing techniques by running offline simulations based on page fault records. The records are collected via executing the OpenMP NAS parallel benchmarks [16] and an OpenMP LINPACK (LPK) benchmark using Intel Cluster OpenMP (CLOMP) [1].

The rest of this paper is organized in seven sections. In section II, we briefly introduce background knowledge including parallel regions and memory consistency models of sDSM systems. Related work on sDSM prefetch techniques

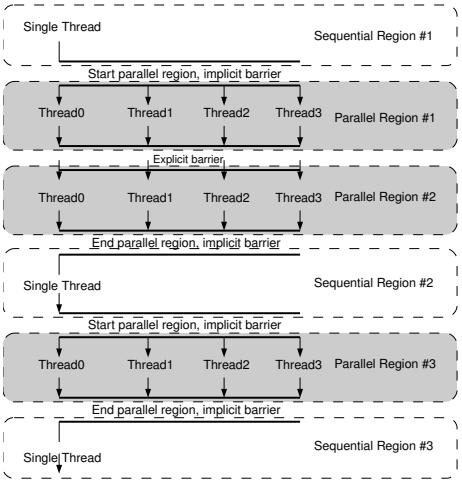

Fig. 1. Illustration of regions in an OpenMP parallel program.

is in section III. In section IV, our new ReP techniques are described in detail, after first noting some of the limitations of current prefetch techniques and analyzing an OpenMP LINPACK program. Simulations are set up in section V and results are presented in section VI. Conclusions are drawn in section VII and future work is briefly described in section VIII.

II. Background

A. Regions of a Parallel Program

A major challenge for shared memory programing is determining what data will be manipulated by which process/thread and how this is coordinated between the processes/threads. Synchronization operations are used to not only exchange data updates and write notices, but also keep the consistency of the data. Often a parallel program is decomposed into several regions separated by global synchronization points in a fork/join programming model. Fig. 1 illustrates this for an OpenMP parallel program.

In Fig. 1, parallel regions can be distinguished by an explicit barrier within a fork-join section, such as parallel region #1 and #2. Moreover, a parallel region can also be distinguished by fork-join operations, such as parallel region #3. The remaining regions, which have only one executing thread, are sequential regions. The CLOMP runtime generates an unique ID for each region (parallel or sequential).

These regions are often enclosed in a loop, which will cause each region to be executed multiple times. The executions of the same region will have their memory access patterns determined by the loop index.

B. sDSM Memory Consistency Model

The majority of sDSM systems are page-based [17], [18], [1]. These partition and manage the globally addressable memory into pages. In addition, page-based sDSM systems can be categorized into two classes, "home-based" and "homeless". TreadMarks and CLOMP are homeless sDSM systems, while JIAJIA, Omni/Danui, Omni/SCASH, Delphi are home-based sDSM systems.

For home-based sDSMs, each page in the shared address space is assigned a *home* where the *master copy* of the page is maintained. The status of the page at its home node is what defines the most "up-to-date" view of that page. Global synchronizations involve all threads sending their page modifications (*diffs*) to the relevant home threads. In this way, 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.

In contrast, homeless sDSM systems [1] do not use page homes. Instead, changes made to a page by other nodes are patched into the local view as required. At a synchronization point, records are made of what pages have been modified since the last synchronization point, and this information is communicated to all 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. Using CLOMP as an example, the actual page modifications are only obtained if and when they are required. This may require the requesting thread to retrieve modifications from multiple other threads.

Besides the difference that a page has a home or not, for both home-based and home-less sDSMs, i) write notices are sent and invalidations are performed at synchronization points, ii) the actual data retrieving happens only when the invalid page is accessed, which avoids unnecessary data transfers, and iii) data fetching to keep memory consistent is the major overhead [7], [8].

III. Related Work

The most relevant existing prefetch techniques include Dynamic Aggregation technique [12], B+ [13], Adaptive++ techniques [11], and third-order DFCM [15].

A. Dynamic Aggregation Technique

The Dynamic Aggregation technique [12] records all page access faults as a fault sequence list, that is divided into multiple groups with a pre-defined group size.

After a global synchronization point, when the first access fault occurs to any page in a group, prefetches are issued for all other pages in that group. In order to record the page fault history, only the faulting page is set to valid after the requested data arrives. After the group is fetched, it is freed. All prefetches need to be guaranteed to arrive before the next synchronization point.

At the next synchronization point, groups are re-calculated based on the access faults experienced by the processor prior to the synchronization point.

B. B+ Technique

B+ [13] is an invalidation-driven prefetch technique, which uses page invalidation to guide prefetching. The B+ technique assumes that a page that has been invalidated within a global

synchronization operation will likely be referenced again in the near future.

Thus, the B+ technique prefetches diffs for each of the pages invalidated at synchronization points. Prefetches are issued right after synchronization points.

C. Adaptive++ Technique

In [11], Adaptive++ technique is developed, which relies on two recorded page fault lists and two modes of operation to predict which pages to prefetch. These two page fault lists are maintained for the previous two regions. Then, during the current barrier, the similarity between the two lists is calculated, and the page fault list for the previous region (p_list) will be chosen if the similarity is greater than 50%. Otherwise, the list for the "before previous" region (bp_list) will be chosen.

The first mode is named the *repeated-phase mode*. In the current barrier, p pre-defined pages from the chosen list (starting from the first page) will be prefetched. Post the barrier, at each page fault, if the page is in the chosen list, the q pre-defined pages following the faulting page from the chosen list will be prefetched[1].

The second mode is named the *repeated-stride mode*. The most frequent page fault stride of the chosen list is used to determine the pages to prefetch in the next phase. Post the current barrier, at each page fault, if the faulting page is in the chosen list, the next q pages with a multiple of the most frequent stride from the faulting page are prefetched.

The decision of which mode to use for the next phase is made during the barrier. If the repeated-phase mode is used for the last region, the efficiency of previously issued prefetches is calculated, using page fault information collected during the execution of the previous regions. If the repeated-phase mode is not used for the last region, the efficiency of what would be issued by this mode is calculated. If the efficiency of the repeated-phase mode is greater than the frequency of the most common stride for the chosen list, the repeated-phase mode is chosen; otherwise the repeated-stride mode is chosen. If neither is greater than 50%, prefetching is avoided until the next barrier.

D. Third Order Differential Finite Context Method

The third-order differential finite context method (TODFCM) is used to predict the most likely page to be accessed next for the Delphi sDSM system [15]. A predictor that continuously monitors all misses to globally shared memory is implemented for TODFCM. For any three consecutive page misses, the predictor records the page number of the next miss in a hash table. During a prediction, a table lookup determines which page miss followed the last time the predictor encountered the same three most recent misses. The predicted page needs to be prefetched before the next actual page miss.

The predictor contains two levels of records. The first level retains the page numbers of the three most recent misses. However, only the most recent page number is stored as an absolute value. The remaining values are the strides between consecutive page numbers. The second level is a hash table which stores the target stride to calculate the next possible page. The entry of second level can be calculated by a given hash operation on the strides stored in first level. The records will be updated when the prediction is not correct. The target stride is replaced by the new stride and the corresponding first level records are updated with the new stride as well.

[1]In [11], p and q are set to 24 and 4 respectively.

IV. Region-Based Prefetch Techniques

In this section, we will first discuss the limitations of related prefetch techniques and analyze an example parallel program. Then, we will propose a set of Region-Based Prefetch (ReP) techniques.

A. Limitations of Current Prefetch Techniques for sDSM Systems

There are some obvious limitations to the above prefetch techniques. The Dynamic Aggregation technique assumes that the page faults that occurred in the current parallel region will occur again in the consecutive parallel region. The B+ technique has a similar assumption. This is obviously not suitable for most parallel applications.

The Adaptive++ technique assumes that a region will experience the same page access strides (same fault pages) as either the previous region or the region before that. Thus, Adaptive++ improves Dynamic Aggregation and the B+ techniques by having page fault records for the two most previous regions and choosing one as target region for prediction, but it is still not adequate for complex parallel applications with multiple parallel regions.

TODFCM is a more generic prefetch technique; however, it does not consider the characteristics of the sDSM memory consistency models. Furthermore, as it only predicts one page faults at a time, only one page can be prefetched in advance, which limits the overlap of computation and communication. Some attempts have been done by Speight *et al.* to reduce the number of network communications via prefetching multiple pages at one time instead of only one page. However, these attempts resulted in a significant reduction in the prefetch efficiency ($\sim$45% reduction for 8 pages and $\sim$25% reduction for 4 pages) [15].

B. Parallel Application Example

To understand better the execution paging behavior of (parallel) regions, we analyze the LINPACK (LPK) OpenMP benchmark as a parallel application example. [2]

Fig. 2 shows pseudo code for the parallelized section of the LINPACK OpenMP benchmark.

In this scheme, the parallel region is re-executed several times. The number of pages accessed in each iteration is decreased, and not all pages accessed in the current execution of the region will be accessed again in the next execution.

[2]This OpenMP LINPACK benchmark is available at http://cs.anu.edu.au/$\sim$Jie.Cai.

```
/* A is a column-major n x n matrix,
   nb is the blocking factor */

for (j = 0; j < n; j += nb) {
  /* region 1 -- sequential */
  read/write(A[j:n-1, 0:n-1]);
  #pragma omp parallel for default(shared) private(jv,jk)
  /* region 2 -- parallel */
  for (jh = j+nb; jh < n; jh++) {
    for (jv = j+nb; jv < n; jv++) {
      for (jk = j; jk < j+nb; jk++){
        read(A[jk,jv]);
        read(A[jh,jk]);
        write(A[jh,jv]);
      }
    }
  } /*end of region 2*/
}
```

Fig. 2. Pseudo code of a LINPACK OpenMP benchmark for a $n \times n$ column-major matrix A with blocking factor nb.

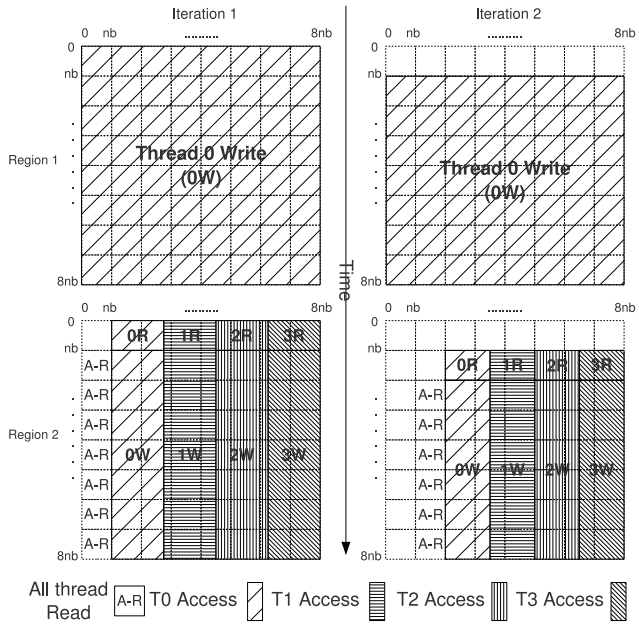

Fig. 3. Illustration of the paging behavior of the LINPACK benchmark for two threads over two iterations.

However, the access pattern changes in a regular fashion. Fig. 3 illustrates the paging behavior of the LINPACK benchmark for four threads over two iterations. In order to simplify the scenario, we set $n = 8nb$ in Fig. 3.

All write operations to the matrix explore both temporal and spatial data locality. All read operations explore only spatial data locality. However, as the memory of matrix A is allocated contiguously, both write and read operations performed by one thread will explore multiple spatial localities (different stride patterns). The stride along rows is 1 page, and the stride along column is $\frac{n}{SystemPageSize}$ pages. However, when $\frac{n}{nb}$ is large, the dominant stride of the application will be 1 page.

While this is not an optimal LINPACK implementation for cluster-enabled OpenMP systems, it provides an interesting pattern of page accesses to test prefetch techniques.

Based on observations of the above LINPACK program and the NPB-OMP suite, we found that these applications exhibit the three major types of behavior:

1) A region executed previously is likely to be executed again in the near future.
2) The page accesses will either show good temporal or spatial locality. In other words, either faulting pages or the strides between the consecutive page faults in an execution of a region are likely to be repeated in the future execution of the same region.
3) Executions of the same region usually exhibit the most similar paging behavior if they are temporally proximate; i.e. in the two previous executions of that region.

In order to address the limitations of those existing sDSM prefetch techniques and fulfill the observations for paging and region execution behavior of parallel applications, we designed two Region-Based Prefetch (ReP) techniques. The first ReP technique only considers the temporal paging behavior, and the second ReP technique addresses all observations.

C. Calculation of Metrics

Before describing ReP techniques, we briefly introduce some metrics which will be used for ReP techniques.

Similarity: the similarity of two page fault lists ($l1$ and $l2$) is calculated as follows.

$$S_{l2}^{l1} = \frac{N_{same}}{N_{l1}} \tag{1}$$

In Equation (1), S_{l2}^{l1} denotes the ratio of N_{same} against N_{l1}. N_{same} stands for number of fault pages belonging to both list, and N_{l1} stand for the number of page faults in l1. Similarly, S_{l1}^{l2} can be calculated. A similarity of greater than 50% means both S_{l2}^{l1} and S_{l1}^{l2} need to be greater than 50%.

Efficiency: this is the ratio of prefetches which were useful. Prefetch techniques usually need to improve this metric to avoid unnecessary prefetches and to eliminate the associated overhead. It can be calculated as follows.

$$E = \frac{N_u}{N_p} \tag{2}$$

In Equation (2), E stands for efficiency, while N_u and N_p denote the number of useful pages, and number of prefetched pages. A *useful* prefetch is a page prefetch issued before the actual access to that page within an execution of a region.[3]

Frequency: it represents how often a stride appears in a page fault list. The most common stride frequency of a given page fault list can be calculated as follows.

$$F_c = \frac{N_c}{N_s} \tag{3}$$

In Equation (3), F_c stands for frequency of the most common stride (c). N_c denotes the number that the most common stride (c) appears in a give page fault list, and $N_s = N_l - 1$ denotes the number of stride in the list.

[3]When a prefetch is not useful for the next execution of a region, it may still be useful for a later execution if the page is not invalidated during the period. However, this is not counted as a *useful* page in the following simulations.

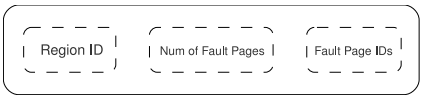

Fig. 4. The page fault record entry for TReP and HReP prefetch techniques.

D. The Temporal ReP (TReP) Technique

Only the temporal paging behavior is considered in the Temporal ReP (TReP) technique.

All experienced page faults are recorded at region-basis for the TReP predictor. Each record entry contains the region ID, number of page faults occurred in the region, and fault page IDs, as shown in Fig. 4.

On the first access to an invalid page after a barrier, the TReP predictor will look-up the previous executions of the current region ID in the records. If there are at least two previous executions, the TReP will issue prefetches for this region. Otherwise, the TReP will not do any page prefetch operations in this region.

To issue prefetches, TReP treat the two recent executions of the current region as two page fault lists. If the two lists are "highly similar", the application is deemed to show good temporal locality, and the whole list of pages for the most recent previous execution of the current region is prefetched. Otherwise, TReP will not prefetch any pages for the current region. The term "highly similar" means that the similarity of these two lists is above a pre-defined threshold (this will be defined and examined in section VI).

E. The Hybrid ReP (HReP) Technique

Based on TReP, we designed a more complex prefetch technique to target both temporal and spatial data locality and all observations listed in section IV-B.

This prefetch mechanism firstly verifies whether the current region has been executed previously. If the current region was executed previously (even only once), the predictor will predicts possible pages that will be accessed in the region by utilizing a hybrid prefetch technique combining TReP and Adaptive++. Therefore, we name it the Hybrid ReP (HReP) technique. The details of the HReP prefetch mechanism is as follows.

In the HReP predictor (Fig. 5), there are totally three prefetch modes, **whole-phase prefetch mode**, **repeated-phase prefetch mode**, and **repeated-stride prefetch mode**. The whole-phase prefetch mode utilizes the TReP technique, while the repeated-phase mode and repeated-stride mode utilize the prefetch mechanisms from the Adaptive++ technique (refer to section III-C). Three paging behaviors were considered: full temporal locality (whole-phase), partial temporal locality (repeated-phase) and spatial locality (repeated-stride).

When p_list and bp_list (refer to section III-C and Fig. 5) are "highly similar", whole-mode is used. Otherwise, either the repeated-phase or the repeated-stride mode is used. Similar to the Adaptive++ technique, whether picking the repeated-phase or the repeated-stride mode depends on the paging behavior of the chosen list. When the efficiency of the prefetches that the repeated-phase mode would have issued for the last execution of this region is greater than or equal to the most common

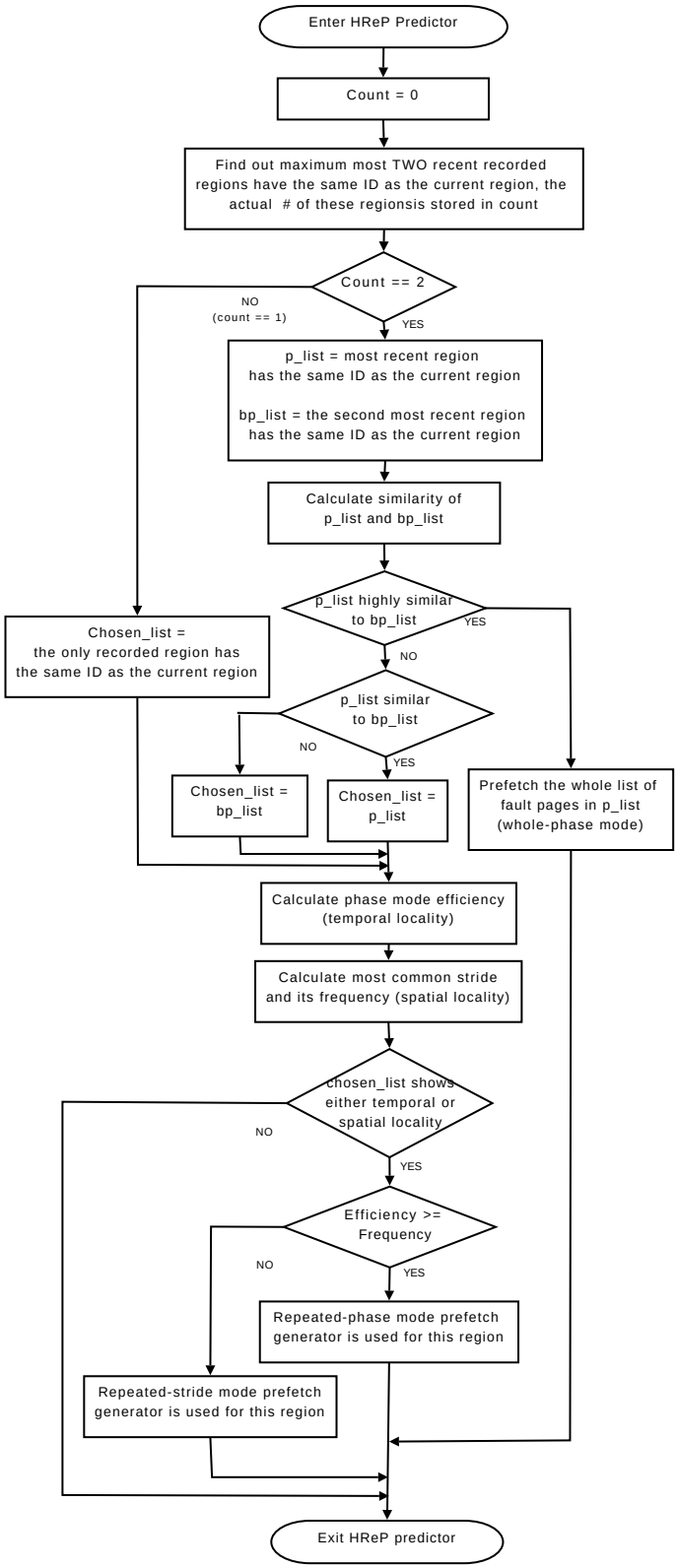

Fig. 5. A flowchart of the HReP predictor.

stride frequency (temporal locality is more dominant than spatial locality), the repeated-phase mode is picked. Otherwise, the repeated-stride mode is picked. If the chosen list does not show either good temporal or spatial locality, the HReP will not issue any prefetches for this region. Good temporal or spatial locality is determined by whether the efficiency or the frequency is higher than a pre-defined value; this value will be determined empirically in section VI.

Although the HReP predictor is developed based on both TReP and Adaptive++ techniques, there are a few differences which improve the prediction compared to the Adaptive++.

Firstly, p_list and bp_list are determined from the first and second most recent executions of the region *with same region ID* as the current region (in Adaptive++, they are determined from the executions of the first and second most recent regions). Since p_list and bp_list are determined from the two most recent executions of the region with the same ID, it is much easier to predict the page faulting pattern for the same region.

Secondly, the determination of the chosen list is different. If only one previous region is found having the same ID as the current region, p_list will be chosen.

Finally, if p_list is "highly similar" to bp_list, the HReP will issue prefetches immediately for all the fault pages of p_list at once, then it exits the HReP predictor. This is the whole-phase prefetch mode.

We will evaluate both the TReP and the HReP techniques using offline simulation.

V. Simulation Setup

According to a comparison of B+, Dynamic Aggregation and Adaptive++, Adaptive++ performs better than other two techniques in terms of efficiency [11]. Therefore, we will only simulate and compare four different sDSM techniques: TReP, HReP, Adaptive++ and TODFCM. Moreover, in [15], evaluation of TODFCM showed that prefetch efficiency is decreased by ∼45% and ∼25% when 8 pages and 4 pages are prefetched at each fault. Therefore, in the simulation for TODFCM, only one page is prefetched at each fault. For repeated-phase mode and repeated-stride mode of both Adaptive++ and HReP, 4 pages will be prefetched at each fault.

A. Offline Page Fault Records

The four different sDSM prefetch techniques have been implemented and evaluated in a offline simulator. The LINPACK benchmark analyzed in section IV-B ($n = 2048$ with $nb = 64$ and $nb = 16$) and some of class A OpenMP NAS Parallel Benchmarks (OMP-NPB) were chosen to generate page fault records on a per thread and per region basis using CLOMP. These records were then used as inputs for the offline simulations.

B. Pre-defined Thresholds for TReP and HReP

The thresholds used in TReP and HReP are: the degrees of similarity to identify "highly similar" and "similar" lists and the degrees of efficiency and frequency to identify whether a region's execution shows good temporal or spatial data locality. They will be defined in this section.

TABLE I
Threshold effects of TReP and HReP for LINPACK benchmark.

Tech	highly similar (%)	2 threads E (%)	2 threads N_p (x1000)	4 threads E (%)	4 threads N_p (x1000)	8 threads E (%)	8 threads N_p (x1000)
TReP	70	87.5	83	84.3	90	87.9	75
	80	88.0	80	88.2	86	91.2	72
	90	90.6	51	90.9	59	91.5	69
HReP	70	83.2	97	81.2	142	80.8	150
	80	85.0	97	81.8	139	81.8	148
	90	85.2	91	82.3	130	82.1	146

Table I shows the effect of different values for the similarity of "highly similar" lists for LINPACK benchmark ($n = 2048$ and $nb = 64$). The results show that both TReP and HReP have an increasing efficiency (E) with increasing threshold for "highly similar" lists. However, the number of prefetches issued (N_p) dropped with increasing threshold. A similar effect is also observed for NPB-OMP benchmarks. Therefore, in order to achieve a balance between the number of prefetches and its efficiency, we chose 80%.

From tests using the LINPACK and NPB-OMP benchmarks, we found that the the optimal similarity threshold for two "similar" lists is 50%. The optimal efficiency and frequency thresholds, indicating good temporal and spatial data locality, were found to be both 50%. These results are similar to what been tested and chosen by Bianchini *et al.* in [11] and Lee *et al.* in [19].

VI. Results and Discussions

In this section, the offline simulation results will be discussed in four aspects: reduction of network communications, prefetch efficiency, prefetch coverage, effective page miss reduction, and a breakdown analysis for the HReP technique.

In general, the fewer useless prefetches (high efficiency) and the fewer subsequent page faults (high coverage), the more effective is the prefetch technique. These metrics correspond to the well-known metrics of precision and recall in text retrieval. The effective page miss reduction is a combination of these metrics corresponding to the impact on prefetch and page fault overhead on execution time.

A. Reduction of Network Communications

As the TReP technique only prefetches once per region execution, it maximally reduces number of network transfers required to serve page misses. HReP is not quite as good, because the repeated-phase and repeated-stride modes are used for runtime prefetch as well. HReP can achieve the same number of reduction in network accesses as TReP only when whole-phase mode is used; and it would reduce network communications by a factor of q when either repeated-phase or repeated-stride mode is used. By contrast, Adaptive++ has the same number of network communications as the worse case for HReP. TODFCM cannot reduce any network communications, as it prefetches a single page at a time.

B. Efficiency

Table II shows simulation results using a different number of threads for the LINPACK and NPB-OMP benchmarks. The total number of page faults (N_f) and the number of prefetches issued (N_p) are presented in thousands, and the number of useful prefetched pages is presented as a ratio to N_f. The efficiency of each prefetch technique is calculated based on Equation (2).

As can be seen from Table II, Adaptive++ maintains prefetch efficiency for LINPACK as the thread number varies; however, it decreases from ∼81% at $nb = 16$ to ∼76% at $nb = 64$. A similar trend is observed for both TReP and HReP as well. Recalling the LINPACK analysis in section IV-B, the memory region accessed at each iteration of the parallel region changes more with increasing block size, thus reducing temporal locality. On the contrary, TODFCM shows a roughly stable prefetch efficiency (∼99%) as the block size is changed. This is because TODFCM only issues prefetch when the same consecutive three faults appears, which avoids useless prefetches in this case.

However, for TReP, the prefetch efficiency slightly increases with increasing number of threads. This is because, with more threads employed, the data is partitioned into finer chunks, and the memory access pattern changes less each iteration. In another words, increasing the number of threads improves temporal locality. On the other hand, HReP shows a slightly decreasing prefetch efficiency of the LINPACK benchmark as more threads are employed. This is because, with a finer grain data distribution, exploiting spatial locality becomes more complex. The portion of consecutive faults with a stride $\frac{n}{SystemPageSize}$ becomes more significant, resulting in an increase in the number of useless prefetches.

For the NPB-OMP benchmarks, the prefetch efficiency of Adaptive++ shows a large variance. Adaptive++ shows good efficiency, 98.1% and 94.7%, for FT and IS respectively. However, less than 45% efficiency is achieved for all other benchmarks. The major reason for this is that its assumptions, namely a consecutive or alternating region repeat pattern, do not hold for the NPB-OMP benchmarks.

In contrast with Adaptive++, the TODFCM prefetch technique shows good prefetch efficiency, around 87.5% to ∼100%, for all NPB-OMP benchmarks except CG. For CG, a ∼70% efficiency is observed, due to the paging behavior of CG being irregular due to sparse matrix access [20]. This breaks TODFCM's assumption of a regular stride.

As TReP will only prefetch when the most recent two executions of the current region are "highly similar", it has achieved very good efficiency for all NPB-OMP benchmarks. TReP shows greater than 99% efficiency for IS, SP, BT and LU, and greater than 96% and 92% efficiency for FT and CG respectively.

HReP is very effective on FT, IS, and BT, with efficiency greater than 96.8%. For SP, the achieved efficiency is 94.1%, 91.9%, and 96.5% for 2 threads, 4 threads and 8 threads respectively. HReP shows reasonable efficiency on CG, from 82.8% to 92.1% for differing number of threads. HReP shows an 88.8% efficiency for LU on 2 threads, and a dramatic decrease to 56.1% efficiency on 8 threads. The reason is that LU benchmark utilizes a lot of flush and lock synchronizations, which will result in the same page faulting multiple times in a region, breaking the repeated-stride mode assumptions.

Comparing the different prefetch techniques, we can find that Adaptive++ shows the worst efficiency for all benchmarks. TReP shows the best efficiency on IS, CG, SP, BT, and LU, while TODFCM shows the best efficiency on FT and LINPACK. Nevertheless, TReP, HReP and TODFCM are quite comparable with each other.

C. Prefetch Coverage

The prefetch coverage corresponds to the 'hit rate' of the pages that would have otherwise faulted in the absence of prefetch. A major observation from Table II is that the coverage (N_u/N_f) of TReP and HReP are much higher than both Adaptive++ and TODFCM.

On average, TReP shows a 46% and 35% better coverage compared Adaptive++ and TODFCM respectively. HReP has 60% and 48% better coverage compared to Adaptive++ and TODFCM respectively, and has the best coverage overall. This is largely due to its use of the *repeated-stride* mode, enabling it to take advantage of spatial locality when page faults, which are not predictable by temporal locality, occur.

D. Effective Miss Rate Reduction

The main objective of an sDSM prefetch technique is to effectively reduce the number of page misses. This is equivalent to the number of effective prefetches, which can be defined using Equation (4).

$$N_e = N_u - (N_p - N_u) \tag{4}$$

N_e stands for the number of effective prefetches, $N_p - N_u$ stands for the number of useless prefetches. This definition reflects the (worst-case) scenario where the cost of prefetching a page is equivalent to the cost of servicing a page fault.

Subsequently, we can calculate the effective miss rate reduction via Equation (5), in which R_{mr} stands for the effective miss rate reduction, N_f stand for the number of total fault pages.

$$R_{mr} = \frac{N_e}{N_f} \tag{5}$$

The effective miss rate reduction based on total page faults (see Table II) is shown in Fig. 6.

This shows that Adaptive++ effectively reduces 24% of the page misses for LINPACK ($nb = 64$) at 2 threads and maintains it with less than 4% difference for 4 and 8 threads. It effectively reduced ∼32% of the page misses for LINPACK benchmark $nb = 16$. ∼50% of the page misses are effectively reduced for IS benchmark, and ∼4% of the page misses are reduced for FT. However, for the other benchmarks (CG, SP, BT and LU), Adaptive++ shows a negative value for the effectively reduced page miss rate, which means that the program execution may slow down from using Adaptive++.

TABLE II
SIMULATION RESULTS FOR ADAPTIVE++, TODFCM (1 PAGE), TREP, AND HREP TECHNIQUES.

Benchmarks	Faults	Adaptive++			TODFCM (1 page)			TReP			HReP		
	(N_f) ($\times$1000)	N_p ($\times$1000)	N_u/N_f (%)	E (%)	N_p ($\times$1000)	N_u/N_f (%)	E (%)	N_p ($\times$1000)	N_u/N_f (%)	E (%)	N_p ($\times$1000)	N_u/N_f (%)	E (%)
2 threads													
LPK ($nb = 64$)	104	56	41.3	76.2	23	21.9	99.5	80	67.5	88.0	97	79.2	85.0
LPK ($nb = 16$)	412	239	46.6	80.6	122	29.4	99.7	392	92.0	96.8	407	94.5	95.8
FT	235	14	5.7	98.1	92	39.1	100.0	101	42.9	100.0	148	62.7	99.6
IS	12	6	46.2	94.7	3	28.2	99.5	6	48.7	99.8	7	56.4	99.2
CG	37	7	8.4	41.8	16	28.0	64.5	32	81.7	94.7	37	90.8	92.1
SP	1709	23	0.5	39.8	722	41.7	98.8	856	50.0	99.8	1608	88.6	94.1
BT	997	7	0.2	30.3	432	43.0	99.1	844	84.6	99.9	970	95.7	98.3
LU	3288	1469	3.9	8.8	782	22.0	92.3	2692	75.4	92.0	3084	83.3	88.8
Average			19.1	58.8		31.7	94.2		67.9	96.4		81.4	94.1
4 threads													
LPK ($nb = 64$)	164	74	34.9	77.6	30	18.3	99.5	86	46.6	88.2	139	69.5	81.8
LPK ($nb = 16$)	628	334	42.8	80.7	170	26.9	99.3	593	89.8	95.2	619	92.7	94.1
FT	357	22	5.8	93.7	139	39.0	99.9	184	49.9	96.6	230	62.6	96.8
IS	25	14	53.9	95.0	9	33.5	99.2	15	58.7	99.8	17	65.7	99.2
CG	108	30	10.7	38.6	43	30.3	75.2	75	66.9	95.5	105	87.7	89.6
SP	3103	104	1.0	28.5	1319	41.7	98.1	2303	74.0	99.7	3138	92.9	91.9
BT	1743	52	1.1	36.8	738	41.7	98.4	1604	91.7	99.6	1710	97.3	99.1
LU	5701	1769	5.8	18.6	1124	18.2	92.3	2332	40.7	99.6	4343	65.7	86.2
Average			19.5	58.7		31.2	95.2		64.8	96.8		79.3	92.3
8 threads													
LPK ($nb = 64$)	211	73	26.8	77.4	34	16.1	98.7	72	30.9	91.2	148	57.3	81.8
LPK ($nb = 16$)	760	333	35.7	81.6	170	22.0	98.9	679	82.9	92.8	743	88.2	90.3
FT	423	27	5.7	87.4	163	38.6	99.8	227	52.9	98.3	270	62.0	97.1
IS	48	31	60.0	94.2	18	37.5	99.4	32	66.3	99.6	36	73.9	99.0
CG	275	45	3.7	22.5	105	26.7	69.8	144	48.1	91.8	269	81.0	82.8
SP	5122	199	0.9	24.4	1360	25.2	94.9	4431	85.9	99.3	5096	96.0	96.5
BT	2747	123	2.1	45.8	1099	39.2	97.9	2627	95.1	99.4	2716	97.9	99.0
LU	9249	3320	5.4	14.9	1670	15.3	84.7	3095	33.3	99.5	7935	48.1	56.1
Average			17.5	56.0		27.6	93.0		61.9	96.5		75.6	87.8

TODFCM effectively reduces ∼18% and ∼28% of the page misses for LINPACK with $nb = 64$ and $nb = 16$ respectively. Between 12% and 40% page misses are effectively reduced for the NPB-OMP benchmarks. Although TODFCM shows best efficiency for FT and LINPACK, and comparable efficiency for IS, SP, BT, and LU benchmarks in Table II, the number of page misses effectively reduced is very much less than TReP and HReP for almost all cases, except for HReP on LU with 8 threads.

TReP effectively reduces more page misses than Adaptive++ and TODFCM for all benchmarks. 58.4% to 28.0% and 89.0% to 76.0% of page misses are effectively reduced for LINPACK ($nb = 64$) and LINPACK ($nb = 16$) respectively. ∼40% to ∼95% page misses are effectively reduced for NPB-OMP benchmarks.

HReP effectively reduces the page miss rate the most for most benchmarks, except for LU with 8 threads. 65.2% to 44.5% of the page misses are effectively removed for LINPACK benchmark with $nb = 64$, and 90.0% to 79.0% of the page misses are effectively removed for LINPACK with $nb = 16$. This observation on effectively reduced page miss rate reflects on the HReP efficiency observed for LINPACK benchmarks in the above section. Moreover, ∼56% to ∼97% of the page misses are effectively reduced for NPB-OMP benchmarks, except for LU with 8 threads. To analyze the reason, we will break down prefetches issued by HReP to find out what is the contribution from different prefetch modes in next section.

On average, TReP effectively reduces page misses 53% and 34% more than does Adaptive++ and TODFCM respectively. HReP technique effectively reduces 62% and 43% more page misses than Adaptive++ and TODFCM respectively.

E. Prefetch Mode Usability Analysis for HReP

Table III shows the number of prefetches issued by different modes and using different chosen lists in the HReP technique.

In Table III, *Tot Pref* stands for total prefetches issued for the benchmark. *W-phase*, *R-phase*, and *R-stride* stand for the number of prefetches issued by the whole-phase prefetch mode, the repeated-phase mode, and the repeated-stride mode respectively (refer to section IV-E). N_{p_l} and N_{bp_l} stands for the number of prefetches issued by using p_list and bp_list as the chosen list respectively.

Except the total prefetches, all other data is presented as a ratio of the total prefetches. For most benchmarks, the *W-phase* is the dominant part of the total prefetches, except LU and LINPACK with $nb = 64$ on 8 threads. *R-phase* only contributes less than 12% of the total prefetches for most benchmarks, except CG, LU, LINPACK ($nb = 64$) with 4 and 8 thread. Moreover, *R-stride* contributes more than 30% for FT and SP with 2 threads, and LU and LINPACK with 8 threads.

The contribution to total prefetches by N_{p_l} and N_{bp_l} equals that of *R-phase* and *R-stride*. As shown by Table III, the ability to select the bp_list is necessary for the FT and SP benchmarks. This is because an alternating behavior of page misses occurs for several regions of thread 0, making *bp_list* a better predictor than *p_list*.

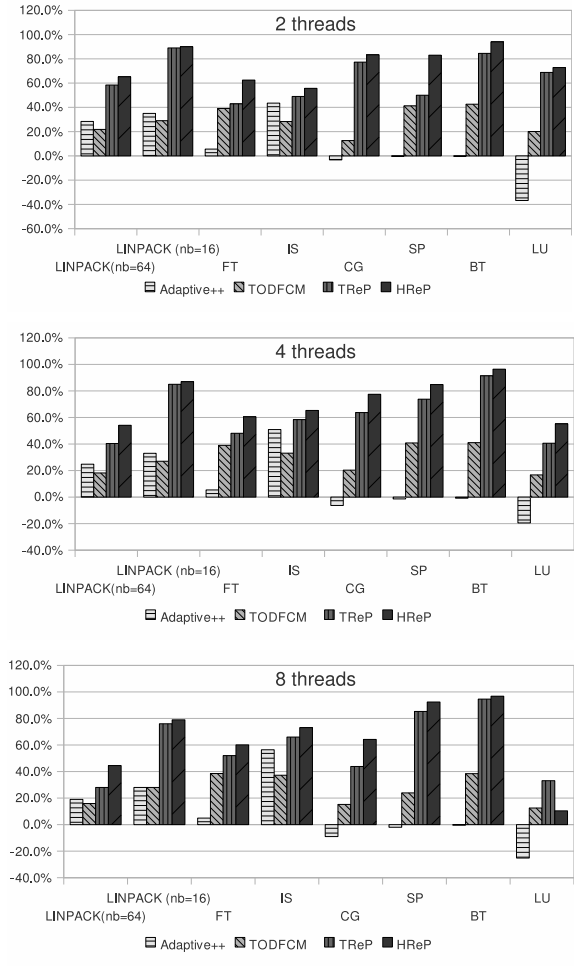

Fig. 6. The effective page miss rate reduction for different prefetch techniques.

LU uses a number of *flush* and lock synchronization operations, which will not start or end a region but does invalidate pages. This will cause the same page to fault multiple times within a region. [4] When data is more finely partitioned (the 8 threads case), this effect becomes more distinct and more frequent. Both temporal locality and spatial locality become worse with an increasing number of threads. Particularly, with 8 threads, the temporal locality becomes worse; however, the most common stride still shows more than a 50% frequency, and so contributes 36.1% of the total prefetches. As a result of this, more useless prefetches are issued.

We have verified this by simulating HReP after removing the page misses caused by flush and lock operations from the collected page fault records of LU. The results are presented in Fig. 7. As we removed some page misses, the number of total faults is reduced, as well as the number of prefetches issued. However, both the efficiency and the effectively reduced page misses increase to ∼99%.

[4]This is because that CLOMP flushes all shared memory instead of a single variable.

TABLE III
BREAKDOWN OF PREFETCHES ISSUED BY DIFFERENT PREFETCH MODES AND CHOSEN LIST DEPLOYED IN HREP.

Benchmarks	Tot Pref	W-phase	R-phase	R-stride	N_{p_l}	N_{bp_l}
2 threads						
LPK ($nb = 64$)	96562	82.4%	8.4%	9.2%	17.4%	0.1%
LPK ($nb = 16$)	406954	96.3%	1.4%	2.3%	3.7%	0.0%
FT	147653	68.2%	0.1%	31.7%	9.2%	22.6%
IS	6612	86.9%	0.1%	13.0%	13.1%	0.0%
CG	36625	87.5%	10.9%	1.6%	9.4%	3.0%
SP	1607569	53.3%	0.9%	45.9%	0.3%	46.3%
BT	970365	87.0%	0.5%	12.6%	0.2%	12.8%
LU	3084373	87.3%	11.9%	0.8%	12.5%	0.1%
4 threads						
LPK ($nb = 64$)	138966	62.2%	18.1%	19.7%	36.8%	1.0%
LPK ($nb = 16$)	618882	95.8%	1.5%	2.8%	4.1%	0.1%
FT	230226	80.0%	0.1%	19.9%	9.0%	11.0%
IS	16880	88.4%	0.2%	11.5%	11.6%	0.0%
CG	105403	71.5%	27.3%	1.2%	25.9%	2.6%
SP	3137665	73.4%	0.8%	25.8%	0.6%	25.9%
BT	1710157	93.8%	0.3%	5.9%	0.5%	5.7%
LU	4343248	53.7%	46.0%	0.3%	35.5%	10.9%
8 threads						
LPK ($nb = 64$)	147558	48.4%	18.3%	33.3%	45.9%	5.4%
LPK ($nb = 16$)	742836	91.4%	5.0%	3.6%	8.1%	0.4%
FT	269677	84.3%	0.1%	15.6%	9.5%	6.2%
IS	36033	89.0%	0.4%	10.6%	11.0%	0.0%
CG	269278	53.5%	45.4%	1.1%	44.0%	2.4%
SP	5095622	87.0%	6.4%	6.7%	1.1%	11.9%
BT	2715739	96.7%	0.3%	2.9%	0.6%	2.7%
LU	7934938	39.0%	24.9%	36.1%	15.9%	45.0%

It should be emphasized here that by using region-based techniques, pages subject to invalidations due to flushes and locks can be predicted in a similar fashion, so such improvements are realizable in a practical sDSM system.

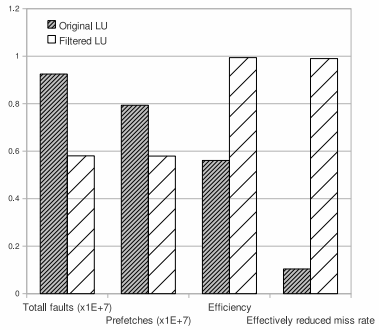

Fig. 7. Prefetch results of HReP for LU after filtered fault pages caused by flushes and locks with 8 threads.

VII. CONCLUSIONS

In this paper, based on the page accessing behavior of distinct parallel regions of a variety of benchmarks, we have designed two region-based prefetch techniques, TReP and HReP, for cluster-enabled OpenMP systems.

TReP utilizes the temporal locality of pages accesses from the current parallel region to prefetch all pages deemed likely to fault in its current execution. The fact that this can be done in bulk promises reduced per-page prefetch overhead, due to the reduction of network communications and the effective utilization of multi-rail interfaces. HReP combines this with the Adaptive++ techniques to prefetch a limited number of

pages each time a page fault occurs. The repeated-phase mode exploits temporal locality in the faulting page (but unlike in Adaptive++, the same region's previous history is used, rather that of previous two regions, which are not necessarily the same as that of the faulting page). The repeated-stride mode uses spatial locality within the current region (and so is the same as in Adaptive++).

We ran offline simulations using page fault records collected by CLOMP with a LINPACK OpenMP benchmark and some NPB-OMP benchmarks to evaluate our proposed prefetch techniques, comparing these with Adaptive++ and TODFCM. On average, TRep and HRep effectively reduced page misses by 63% and 71% for all benchmarks. This represents an improvement of 53% and 34% (TRep) and 62% and 43% (HRep) on Adaptive++ and TODFCM respectively. In terms of efficiency, TRep showed the best efficiency overall, followed closely by TODFCM. The main difference in effective page miss reduction was however due to coverage, with HRep achieving 14% better coverage than TRep, largely due to its exploitation of spatial locality. TRep in turn achieved 35% and 46% better coverage than TODFCM and Adaptive++ respectively.

Pages that are invalidated from locks and flushes, such as in the LU benchmark, cause page misses that cannot be avoided using prefetch techniques. These contributed to the miss rates for all methods, and, in some cases, decreased the efficiency as well. However, they can be predicted using region-based techniques and removed from consideration of prefetching.

VIII. Future Work

Both whole-phase and repeated-phase rely on high temporal locality in page fault lists between different executions of a region. We are investigating techniques that also can make predictions based on any spatial locality between the lists. This involves grouping pages in each list into identifiable "blocks", and identifying how these blocks shift between the two lists.

In the future, we will attempt to implement these techniques in the CLOMP runtime. This will raise several issues. To reduce the storage and overhead of processing the necessary page fault lists, we will look at compressed representations. This is also likely to improve the efficiency of similarity calculations. Furthermore, the offline simulation is time independent and assumes all the prefetched page will be available before the actual access. However, some associated overheads may be introduced for a real implemented system. These issues will be considered during implementation.

Acknowledgment

This work is funded by Australian Research Council Grant LP0669726, ANU CECS Faculty Research Grant, Intel Corporation and Sun Microsystems. Special thanks go to Pete P. Janes, Daniel Robson and Xi Yang for their helpful discussions.

References

[1] J. P. Hoeflinger, "Extending OpenMP to Clusters," *White Paper, Intel Corporation*, 2006.

[2] 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*, 2007, pp. 231–240.

[3] M. Sato, H. Harada, and Y. Ishikawa, "OpenMP Compiler for a Software Distributed Shared Memory System SCASH," in *WOMPAT2000*, Jul. 2000. [Online]. Available: citeseer.ist.psu.edu/sato00openmp.html

[4] M. Sato, S. Satoh, K. Kusano, and Y. Tanaka, "Design of OpenMP Compiler for an SMP Cluster," in *EWOMP '99*, Sep. 1999, pp. 32–39. [Online]. Available: citeseer.ist.psu.edu/sato99design.html

[5] Y.-S. Kee, J.-S. Kim, and S. Ha, "ParADE: An OpenMP Programming Environment for SMP Cluster Systems," in *Proceedings of the ACM/IEEE SC2003 conference on High Performance Networking and Computing*, 2003, p. 6.

[6] R. Yang, J. Cai, A. P. Rendell, and V. Ganesh, "Use of Cluster OpenMP with the Gaussian Quantum Chemistry Code: A Preliminary Performance Analysis," in *LNCS 5568, IWOMP 2009*, 2009, pp. 53–62.

[7] H. J. Wong, J. Cai, A. P. Rendell, and P. E. Strazdins, "Micro-Benchmarks for Cluster OpenMP Implementations: Memory Consistency Costs," in *IWOMP 2008, LNCS 5004*, 2008, pp. 60–70.

[8] J. Cai, A. P. Rendell, P. E. Strazdins, and H. J. Wong, "Performance Model for Cluster-Enabled OpenMP Implementations," in *13th IEEE Asia-Pacific Computer Systems Architecture Conference*, 2008, pp. 1–8.

[9] J. Cai, A. P. Rendell, and P. E. Strazdins, "Non-threaded and Threaded Approaches to Multirail Communication with uDAPL," in *6th IFIP International Conference on Network and Parallel Computing*, 2009, pp. 233–239.

[10] J. Liu, A. Vishnu, and D. K. Panda, "Building Multirail InfiniBand Clusters: MPI-Level Design and Performance Evaluation," in *In Supercomputing*. IEEE Computer Society, 2004, p. 33.

[11] R. Bianchini, R. Pinto, and C. L. Amorim, "Data prefetching for software DSMs," in *ICS '98: Proceedings of the 12th international conference on Supercomputing*, 1998, pp. 385–392.

[12] C. Amza, A. Cox, K. Rajamani, and W. Zwaenepoel, "Tradeoffs between false sharing and aggregation in software distributed shared memory," in *PPoPP '97: Proceedings of the sixth ACM SIGPLAN symposium on Principles and practice of parallel programming*, 1997, pp. 90–99.

[13] R. Bianchini, L. I. Kontothanassis, R. Pinto, M. De Maria, M. Abud, and C. L. Amorim, "Hiding communication latency and coherence overhead in software DSMs," *SIGOPS Oper. Syst. Rev.*, vol. 30, no. 5, pp. 198–209, 1996.

[14] B. Goeman, H. Vandierendonck, and K. de Bosschere, "Differential FCM: increasing value prediction accuracy by improving table usage efficiency," in *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*, 2001, pp. 207–216.

[15] E. Speight and M. Burtscher, "Delphi: Prediction-Based Page Prefetching to Improve the Performance of Shared Virtual Memory Systems," in *In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, 2002, pp. 49–55.

[16] H. Jin, M. Frumkin, and J. Yan, "The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance," *Technical Report: NAS-99-011*, 1999. [Online]. Available: citeseer.ist.psu.edu/408248.html

[17] H. J. Wong and A. P. Rendell, "Integrating Software Distributed Shared Memory and Message Passing Programming," in *Cluster09*, 2009, pp. 1–10.

[18] W. Hu, W. Shi, and Z. Tang, "JIAJIA: A Software DSM System Based on a New Cache Coherence Protocol," in *HPCN Europe '99: Proceedings of the 7th International Conference on High-Performance Computing and Networking*. London, UK: Springer-Verlag, 1999, pp. 463–472.

[19] S.-K. Lee, H.-C. Yun, J. Lee, and S. Maeng, "Adaptive prefetching technique for shared virtual memory," in *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, 2001, pp. 521–526.

[20] J.Cai, A. Rendell, P. Strazdins, and H. Wong, "Predicting Performance of Intel Cluster OpenMP with Code Analysis Method," in *ANU Computer Science Technical Reports, TR-CS-08-03*, 2008.