Region-based Techniques for Modeling and Enhancing Cluster OpenMP Performance

Jie Cai

August 2011

A thesis submitted for the degree of Doctor of Philosophy of the Australian National University
© Jie Cai 2011

This document was produced using \TeX{}, \LaTeX{} and \BIBTeX{}
For my wife, Ruru, who greatly supported my PhD research...

...and my loving parents.
Declaration

I declare that the work in this thesis is entirely my own and that to the best of my knowledge it does not contain any materials previously published or written by another person except where otherwise indicated.

Jie Cai
Acknowledgements

During my PhD, many people have offered kind help and generous support. I would like to thank them and appreciate their help.

Supervisors

Dr. Peter Strazdins for the guidance and advice during my whole doctoral research; Dr. Alistair Rendell for the time we spent together before paper deadlines; Dr. Eric McCreath for the useful comments.

Readers

For reading my thesis and providing valuable feedbacks, thank you. Warren Armstrong, Muhammad Atif, Michael Chapman, Pete Janes, Josh Milthorpe, Peter Strazdins, and Jin Wong.

Computer System Group Members

For the cheerful four years of my PhD, thank you “geeks”. Joseph Anthony, Ting Cao, Elton Tian, Xi Yang, Fangzhou Xiao and more ...

Industry Partners

For their generous financial contribution to support my research. Australian Research Council, Intel, Sun Microsystems (Oracle)

Last but definitely not least

For being so supportive, NCI NF colleagues. Ben Evans, Robin Humble, Judy Jenkinson and David Singleton.
Abstract

Cluster OpenMP enables the use of the OpenMP shared memory programming model on distributed memory cluster environments. Intel has released a cluster OpenMP implementation called Intel Cluster OpenMP (CLOMP). While this offers better programmability than message passing alternatives such as the Message Passing Interface (MPI), such convenience comes with overheads resulting from having to maintain the consistency of underlying shared memory abstractions. CLOMP is no exception. This thesis introduces models for understanding these overheads of cluster OpenMP implementations like CLOMP and proposes techniques for enhancing their performance.

Cluster OpenMP systems are usually implemented using page-based software distributed shared memory (sDSM) systems, which create and maintain virtual global shared memory spaces in pages. A key issue for such system is maintaining the consistency of the shared memory space. This forms a major source of overhead, and it is driven by detecting and servicing page faults.

To investigate and understand these systems, we evaluate their performance with different OpenMP applications, and we also develop a benchmark, called MCBENCH, to characterize the memory consistency costs. Using MCBENCH, we discover that this overhead is proportional to the number of writers to the same shared page and the number of shared pages.

Furthermore, we divide an OpenMP program into separate parallel and serial regions. Based on the regions, we develop two region-based models to rationalize the numbers and types of the page faults and their associated costs to performance. The models highlight the fact that the major overhead is servicing the type of page faults, which requires data (a page or its modifications, known as diffs) to be transferred across a network.

With this understanding, we have developed three region-based prefetch (ReP) techniques based on the execution history of each parallel and sequential region. The first ReP technique (TReP) considers temporal page faulting behaviour between consecutive executions of the same region. The second technique (HReP) considers both the temporal page faulting behaviour between consecutive executions of the same region and the spatial paging behaviour within an execution of a region. The last technique (DReP) utilizes our proposed novel stride-augmented run length encoding (sRLE) method to address the both the temporal and spatial page faulting behaviour between consecutive executions of the same region. These techniques effectively reduce the number of page faults and aggregate data (pages and diffs) into larger transfers, which leverages the network bandwidth provided
by interconnects.

All three ReP techniques are implemented into runtime libraries of CLOMP to enhance its performance. Both the original and the enhanced CLOMP are evaluated using the NAS Parallel Benchmark OpenMP (NPB-OMP) suite, and two LINPACK OpenMP benchmarks on different hardware platforms, including two clusters connected with Ethernet and InfiniBand interconnects. The performance data is quantitatively analyzed and modeled. Also, MCBENCH is used to evaluate the impact of ReP techniques on memory consistency cost.

The evaluation results demonstrate that, on average, CLOMP spends 75% and 55% overall elapsed time of the NPB-OMP benchmarks on Gigabit Ethernet and double data rate InfiniBand network respectively. These ratios of the NPB-OMP benchmarks are reduced effectively by \( \sim 60\% \) and \( \sim 40\% \) after implementing the ReP techniques on to the CLOMP runtime. For the LINPACK benchmarks, with the assistance of sRLE, DReP significantly outperforms the other ReP techniques with effectively reducing 50% and 58% of page fault handling costs on Gigabit Ethernet and InfiniBand networks respectively.
# Contents

Declaration v

Acknowledgements vii

Abstract ix

I Introduction and Background 1

1 Introduction 3

1.1 Motivation 4

1.1.1 Research Objectives 5

1.2 Contributions 6

1.2.1 Performance Evaluation of CLOMP 6

1.2.2 Region-based Performance Models 7

1.2.3 Region-based Prefetch Techniques 8

1.3 Thesis Structure 9

2 Background 11

2.1 OpenMP 12

2.1.1 OpenMP Directives 12

2.1.2 Synchronization Operations 16

2.2 Cluster OpenMP Systems 17

2.2.1 Relaxed Memory Consistency 18

2.2.2 Software Distributed Shared Memory Systems 19

2.2.3 Intel Cluster OpenMP 23

2.2.4 Alternative Approaches to sDSMs 26

2.3 Related Work 29

2.3.1 Performance Models 29

2.3.2 Prefetch Techniques for sDSM Systems 31

2.3.3 Run-Length Encoding Methods 35

2.4 Summary 37

II Performance Issues of Intel Cluster OpenMP 39

3 Performance of Original Intel Cluster OpenMP System 41

3.1 Hardware and Software Setup 42
6 Implementation and Evaluation

6.1 ReP Prefetch Techniques Implementation Issues
   6.1.1 Data Structures
   6.1.2 New Region Notification
   6.1.3 Record Encoding and Flush Filter enabled Decoding
   6.1.4 Prefetch Page Prediction
   6.1.5 Prefetch Request and Event Handling
   6.1.6 Page State Transition
   6.1.7 Garbage Collection Mechanism

6.2 Theoretical Performance of the ReP Enhanced CLOMP

6.3 Performance Evaluation of the ReP Enhanced CLOMP
   6.3.1 MCBENCH
   6.3.2 NPB OpenMP Benchmarks
   6.3.3 LINPACK Benchmarks
   6.3.4 ReP Techniques with Multiple Threads per Process

6.4 Summary

IV Conclusions and Future Work

7 Conclusions and Future Work

7.1 Conclusions
   7.1.1 Performance Evaluation of CLOMP
   7.1.2 SIGSEGV Driven Performance Models
   7.1.3 Performance Enhancement by RePs

7.2 Future Directions
   7.2.1 Performance Evaluation
   7.2.2 Performance Optimizations
   7.2.3 Adapting ReP Techniques to the Latest Technologies
   7.2.4 Potential Use of sRLE

V Appendices

A Algorithms Used in DReP
   A.1 Stride-augmented Run-length Encoding Algorithms
      A.1.1 Algorithm 1: Page Fault Record Reconstruction Step (a)
      A.1.2 Algorithm 2: Page Fault Record Reconstruction Step (b)
      A.1.3 Algorithm 3: Page Fault Record Reconstruction Step (c)
   A.2 Algorithm 4: DReP Predictor
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>OpenMP fork-join multi-threading parallelism mechanism [93]</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>OpenMP parallel directives and associated clauses in C and C++.</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>OpenMP for directives and associated clauses in C and C++.</td>
<td>14</td>
</tr>
<tr>
<td>2.4</td>
<td>An example OpenMP program in C using parallel for directives.</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>OpenMP synchronization directives in C and C++ languages: (a) barrier, and (b) flush.</td>
<td>15</td>
</tr>
<tr>
<td>2.6</td>
<td>OpenMP threadprivate directive in C and C++ languages.</td>
<td>16</td>
</tr>
<tr>
<td>2.7</td>
<td>Processes and threads in CLOMP</td>
<td>23</td>
</tr>
<tr>
<td>2.8</td>
<td>State machine of CLOMP (derived from [47], [38], and experimental observation).</td>
<td>25</td>
</tr>
<tr>
<td>2.9</td>
<td>Illustration of two prefetch modes for Adaptive++ techniques.</td>
<td>34</td>
</tr>
<tr>
<td>3.1</td>
<td>Comparison of performance between native Intel OpenMP and CLOMP on a XE compute node.</td>
<td>45</td>
</tr>
<tr>
<td>3.2</td>
<td>Comparison of performance between native Intel OpenMP and CLOMP on a VAYU compute node.</td>
<td>46</td>
</tr>
<tr>
<td>3.3</td>
<td>Performance of CLOMP on XE with a single thread per compute node.</td>
<td>49</td>
</tr>
<tr>
<td>3.4</td>
<td>Performance of CLOMP on VAYU with a single thread per compute node.</td>
<td>50</td>
</tr>
<tr>
<td>3.5</td>
<td>Performance of CLOMP on XE with multi-threads per compute node.</td>
<td>52</td>
</tr>
<tr>
<td>3.6</td>
<td>Performance of CLOMP on VAYU with multi-threads per compute node.</td>
<td>53</td>
</tr>
<tr>
<td>3.7</td>
<td>MCBENCH – An array of size $a$-bytes is divided into chunks of $c$-bytes. The benchmark consists of Change and Read phases that can be repeated for multiple iterations. Entering the Change phase of the first iteration, the chunks are distributed to the available threads (four in this case) in a round-robin fashion. In the Read phase after the barrier, each thread reads from the chunk that its neighbour had written to. This is followed by a barrier which ends the first iteration. For the subsequent iteration, the chunks to Change are the same as in the previous Read phase. That is, the shifting of the chunk distribution only takes place when moving from the Change to Read phases.</td>
<td>57</td>
</tr>
</tbody>
</table>
3.8 MCBENCH evaluation results of CLOMP on XE with both Ethernet and InfiniBand interconnects: 64KB, 4MB and 8MB array sizes are used in these three figures respectively; comparison among different chunk sizes 4B, 2KB and 4KB is illustrated in each figure for both Ethernet and InfiniBand.

4.1 Illustration of regions in an OpenMP parallel program.

4.2 Schematic illustration of timing breakdown for parallel region using the SDP model.

4.3 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 $R$.

5.1 Pseudo code to demonstrate the memory access patterns of the naive LINPACK OpenMP benchmark implementation for an $n \times n$ column-major matrix $A$ with blocking factor $nb$.

5.2 Naive OpenMP LINPACK program with $n \times n$ matrix: (a) memory access areas for different iterations. (b) page fault areas for different iterations.

5.3 Pseudo code to demonstrate the memory access patterns of the optimized LINPACK OpenMP benchmark implementation for an $n \times n$ column-major matrix $A$ with blocking factor $nb$.

5.4 Optimized OpenMP LINPACK program: (a) memory access areas for different iterations illustrated on a $n \times n$ matrix panel. (b) page fault areas for different iterations illustrated on the $n \times n$ matrix panel.

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

5.6 A flowchart of the HReP predictor.

5.7 Two levels of stride-augmented run-length encoding (sRLE) method: (a) Based on strides between consecutive pages, sorted missed pages are broken into small sub-arrays, and those consecutive pages with the same stride are stored in the same array. (b) The sub-arrays are compressed in to the first level sRLE records in a $(\text{StartPageID}, \text{CommonStride}, \text{RunLength})$ format. (c) Based on the stride between the start pages of consecutive first level sRLE records, they are further compressed into the second level sRLE format, $(\text{FirstLevelRecord}, \text{CommonStride}, \text{RunLength})$ (more details in Section 5.5.1).

5.8 Page fault record of region execution reconstructed via run-length encoding method.
D.4 Threaded multirail benchmark design. . . . . . . . . . . . . . . . . . . 183
D.5 RDMA write latency comparison. . . . . . . . . . . . . . . . . . . . . 184
D.6 Uni-directional multi-port bandwidth. . . . . . . . . . . . . . . . . . 185
D.7 Uni-directional multi-HCA bandwidth. . . . . . . . . . . . . . . . . . 186
D.8 Bi-directional multi-port bandwidth. . . . . . . . . . . . . . . . . . . 187
D.9 Bi-directional multi-HCA bandwidth. . . . . . . . . . . . . . . . . . . 188
D.10 Benchmarks elapsed time breakdown for 512bytes message. . . . . 188
D.11 Benchmarks elapsed time breakdown for 4KB message. . . . . . . . 189
D.12 Different ways to configure a InfiniBand multirail network [62]. . . 190
List of Tables

2.1 OpenMP synchronization operations. ......................................... 17

3.1 Evaluation experimental hardware platforms. .......................... 43

3.2 Sequential elapsed time (sec) of NPB with CLOMP. ..................... 44

3.3 Page faults handling cost (SEGV Cost) of CLOMP for NPB benchmarks as a ratio to corresponding elapsed time with single thread per process on XE. ......................................................... 62

3.4 Page faults handling cost breakdown for CLOMP for class A NPB benchmarks with multiple threads per process on XE. “SEGV” represents the ratio of page faults handling cost to the corresponding elapsed time; “SEGV Lock” in turn represents a ratio of pthread mutex cost within “SEGV”. ......................................................... 63

4.1 Critical path page faults counts for the NPB-OMP benchmarks run using CLOMP ............................................................ 73

4.2 Comparison between observed and estimated speedup for running NPB class A and C on the AMD cluster with CLOMP ...................... 77

4.3 Average relative errors for the predicted NPB speedups evaluated using the critical path and aggregate (f = 0) SDP models and data from Tables 4.2. .............................................................. 78

5.1 Threshold effects of ReP techniques for naive LINPACK benchmark. 98

5.2 Simulation prefetch efficiency (E) and coverage (N_u/N_f) for Adaptive++, TODFCM (1 page), TReP, HReP and DReP techniques. ....... 108

5.3 Breakdown of prefetches issued by different prefetch modes and chosen list deployed in HReP. ..................................................... 109

5.4 Comparison of F-HReP and HReP with the LU benchmark. .......... 109

6.1 Bandwidth and latency measured by the communication layer (CAL) of CLOMP. ............................................................... 121

6.2 ReP techniques prefetch efficiency and coverage for MCBNECH with 4MB array. .............................................................. 123

6.3 Message transfer counts (×1000) comparison between RePs enhanced CLOMP and the original CLOMP for MCBENCH with 4KB chunk . 126

6.4 Message transfer counts (×1000) comparison between RePs enhanced CLOMP and the original CLOMP for MCBENCH with 2KB chunk . 128

6.5 Message transfer counts (×1000) comparison between RePs enhanced CLOMP and the original CLOMP for MCBENCH with 4KB chunk . 129
6.6 Page fault handling costs comparison for BT benchmark among the original CLOMP, the theoretical and the ReP techniques enhanced CLOMP. The computation part of elapsed time is common to all compared items. The page fault handling costs of the original CLOMP is presented in second, and that of others are presented as a reduction ratio (e.g. \( \frac{\text{Orig} - \text{TRep}}{\text{Orig}} \)).

6.7 Page fault handling costs reduction ratio \( \left( \frac{T_{\text{orig segv}} - T_{\text{segv}}}{T_{\text{segv}}} \right) \) comparison for other NPB benchmarks.

6.8 Detailed \( T_{\text{segv}} \) breakdown analysis of the IS Class A Benchmark for the ReP techniques. Overall \( T_{\text{segv}} \) stands for overall CLOMP overhead. “TMK Comm” stands for the communication time spent by TMK for data transfer. “TMK local” stands for the local software overhead of TMK layer. “ReP Comm” stands for the communication time spent on prefetching data. “ReP local” stands for the local software overhead introduced by using the ReP prefetch techniques. \( T_{\text{segv}} \) is presented in seconds and its components are presented as a ratio to the overall \( T_{\text{segv}} \).

6.9 Detailed \( T_{\text{segv}} \) breakdown analysis of the IS Class C Benchmark for the ReP techniques. Overall \( T_{\text{segv}} \) stands for overall CLOMP overhead. “TMK Comm” stands for the communication time spent by TMK for data transfer. “TMK local” stands for the local software overhead of TMK layer. “ReP Comm” stands for the communication time spent on prefetching data. “ReP local” stands for the local software overhead introduced by using the ReP prefetch techniques. \( T_{\text{segv}} \) is presented in seconds and its components are presented as a ratio to the overall \( T_{\text{segv}} \).

6.10 Sequential elapsed time for LINPACK benchmarks.

6.11 Page fault handling costs comparison for LINPACK benchmarks among the original CLOMP, the theoretical and the ReP techniques enhanced CLOMP. The computation part of elapsed time is common to all compared items. The page fault handling costs of the original CLOMP is presented in second, and that of others are presented as a reduction ratio (e.g. \( \frac{\text{Orig} - \text{TRep}}{\text{Orig}} \)).

6.12 Page faults handling cost comparison between DReP and the original CLOMP for the optimized LINPACK benchmark with multiple threads per process. “SEGV” represents the ratio of page faults handling cost to the corresponding elapsed time; “SEGV Lock” in turn represents a ratio of pthread mutex cost within “SEGV”.

132

135

136

137

138

139

142
B.1 $T_{\text{segv,local}}$ (sec) for some NPB-OMP benchmarks with different number of processes. ......................................................... 166
B.2 $N_{\text{total}}^f$ for some NPB-OMP benchmarks with different number of processes. ............................................................. 167
B.3 $T_{\text{segv, total}}^f$ (sec) for LINPACK benchmarks with different number of processes. ............................................................. 167
B.4 $N_{\text{total}}^f$ for LINPACK benchmarks with different number of processes. 167

C.1 Elapsed Time (sec) of some NPB-OMP Benchmarks on one thread. . 170

E.1 Complete bandwidth and latency measured by the communication layer (CAL) of CLOMP on XE. ............................. 194
E.2 Comparison of CAL and OpenMPI: bandwidth and latency measured on XE via GigE. ........................................................... 195
E.3 Comparison of CAL and OpenMPI: bandwidth and latency measured on XE via DDR IB. ........................................................... 195