Lehrstuhl für Rechnertechnik und
Rechnerorganisation
der Technischen Universität München

Data Locality Optimization of Shared Memory Programs on NUMA Architectures Using an Integrated Tool Environment

Jie Tao

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. H. M. Gerndt
Prüfer der Dissertation:	
	1. Univ.-Prof. Dr. A. Bode
	2. Univ.-Prof. Dr. E. Jessen

Die Dissertation wurde am 20.06.2002 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 29.08.2002 angenommen.

Research Report Series
Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR-TUM)
Technische Universität München

http://www.bode.in.tum.de/

Editor: Prof. Dr. A. Bode

Vol. 26

Data Locality Optimization of Shared Memory Programs on NUMA Architectures Using an Integrated Tool Environment

Jie Tao

SHAKER
VERLAG
Aachen 2002

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Tao, Jie:
Data Locality Optimization of Shared Memory Programs on
NUMA Architectures Using an Integrated Tool Environment /
Jie Tao. Aachen : Shaker, 2002
(Research Report Series / Lehrstuhl für Rechnertechnik und Rechner-
organisation (LRR-TUM), Technische Universität München ; Vol. 26)
Zugl.: München, Techn. Univ., Diss., 2002
ISBN 3-8322-0982-4

Copyright Shaker Verlag 2002
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publishers.

Printed in Germany.

ISBN 3-8322-0982-4
ISSN 1432-0169

Shaker Verlag GmbH • P.O. BOX 101818 • D-52018 Aachen
Phone: 0049/2407/9596-0 • Telefax: 0049/2407/9596-9
Internet: www.shaker.de • eMail: info@shaker.de

Abstract

Due to their excellent price-performance ratio, clusters built from commodity nodes have become broadly adopted and increasingly popular as platforms for parallel processing. Among them, the clusters of standard PCs interconnected with high-speed system area networks (SANs) are especially attractive and have been widely established. At the same time, the developments in interconnection technologies also formed the basis for the rise of Non-Uniform Memory Access (NUMA) architectures, i.e. systems with physically distributed memories, but with a global address space allowing an efficient but non-uniform access to any memory location in the system. These kinds of systems, especially when offered as non cache coherent NUMA for loosely coupled commodity architectures, can easily be implemented in a straightforward manner without major hardware efforts. They form a favorable architectural tradeoff by combining the scalability and cost–effectiveness of standard clusters with a shared memory support close to symmetric multiprocessors.

The non-uniform memory access characteristic, however, introduces a distinction between local and remote memory causing different memory access latencies. In systems with such characteristics, a remote memory access can take up to an order of magnitude longer than a local one. This leads to the fact that many shared memory applications initially do not achieve a good parallel speedup when running on NUMA-like architectures due to excessive remote memory accesses.

This thesis targets such inefficiency problems of NUMA-based shared memory programs. For this purpose, a comprehensive and integrated tool environment has been built which aims at improving the data locality of running applications, by combining single frameworks enabling both program tuning and runtime manipulation. This environment comprises a low-level data acquisition system, a distributed tool middleware, and a set of performance tools. Based on the hardware monitoring facility, which is capable of observing all memory transactions performed on the interconnection fabric, the data acquisition system provides information about an application's memory access behavior as well as information about e.g. synchronization primitives and address mapping necessary for data placement. This information is then aggregated across the distributed system and made accessible to the tools through an established on-line monitoring interface specification serving as middleware. Tools further process the acquired performance data and use it to steer the execution of programs with a result of an optimization for the runtime data layout.

Currently, two such tools have been implemented: a Data Layout Visualizer (DLV) and an Adaptive Runtime System (ARS). DLV is used to present the monitoring data in an easyto-use fashion allowing a programmer to understand the dynamic behavior of an application and to detect and correct communication hot spots. ARS is used to transparently modify the data layout during the execution of a program without involving the user. Both tools are capable of introducing a better data locality and hence a better performance.

In addition, the tool environment includes an event-driven multiprocessor simulator called SIMT. SIMT was originally designed to supply the monitoring information in case no hardware monitor is available. It has been developed, however, to be fully flexible with respect to the target architectures and can thereby be used as a general tool for system design and performance evaluation of PC clusters with NUMA organization.

The monitoring and tool environment has been evaluated using a set of shared memory applications selected from standard benchmarks and kernels. Most of them show a high improvement in terms of absolute execution time and speedup, proving the feasibility and effectiveness of this monitoring approach. Furthermore, the monitoring approach provides the basic means to ensure interoperability between shared memory as well as existing paradigm independent tools. This strongly contributes to the flexibility of the proposed approach and also enables the seamless integration into existing systems.

Acknowledgments

At this minute I am very excited because I have finished this work. I know at the same time that it would not have been possible without the support of many others. I would like to take this opportunity to express my sincere appreciations of their help.

First of all I would like to thank my advisor, professor Arndt Bode, for providing me with an excellent research environment and giving me the greatest freedom to conduct my research. I am also appreciate of his help in overcoming all the possible financial and administrative issues. In addition, he had always time for me despite his busy schedules.

I would like to thank professor Eike Jessen for taking the time to review this thesis and giving valuable comments and detailed improvements regarding the language.

I would like to express my special gratitude to Dr. Wolfgang Karl, the leader of my research group. He not only directed me to the right way for pursuing my work but also gave me the most considerate help throughout the whole course of this research work. In addition, I especially appreciate his friendship and the time and enthusiasm for helping me to feel at home in Germany.

Besides all my co-workers and the technical and administrative staff at LRR-TUM, I would like to specially mention Dr. Martin Schultz who worked together with me on the same project. His suggestions and good ideas directly contributed to the success of this work. I am especially grateful for his patience and tolerance to my poor German. He also deserves this special thanks for directing me to write English not in a Chinese way.

My stay in Germany has been supported by the German foundation Friedrich-Ebert-Stiftung. It deserves my special thanks.

Finally, I would like to express my deepest gratitude to my parents who have shown great understanding for my impossibility to live around them. I thank my brother and sisters who share my responsibility for taking care of our parents.

Jie Tao
Munich, Germany
June 2002

Contents

1 Motivation **1**
- 1.1 NUMA and its Associated Problems 1
- 1.2 The Approach 2
- 1.3 Contributions 4
- 1.4 Structure of the Thesis 6

2 Establishing a Tool Environment for Locality Optimization: Overview **9**
- 2.1 Challenges of this Approach 9
- 2.2 SMiLE: Building a Sample NUMA Architecture 11
  - 2.2.1 The Scalable Coherent Interface 11
  - 2.2.2 SCI Virtual Memory 12
  - 2.2.3 HAMSTER 13
- 2.3 The Tool Environment on top of SMiLE 14
  - 2.3.1 Hardware Monitoring and Data Acquisition 15
  - 2.3.2 OMIS/OCM as Monitoring Middleware 17
  - 2.3.3 Performance Tools 18
  - 2.3.4 The Multiprocessor Simulator 19
- 2.4 Summary 19

3 On-line Monitoring of Shared Memory Traffic **21**
- 3.1 Challenges 21
- 3.2 The SMiLE Hardware Monitor: Concept 22
  - 3.2.1 SMiLE PC Cluster Architecture and the PCI-SCI Adapter 22
  - 3.2.2 Monitor Hardware Design 24
  - 3.2.3 Working Principles 26
- 3.3 The SMiLE Hardware Monitor: Prototypical Implementation 28
  - 3.3.1 B-Link Interface 29
  - 3.3.2 Counter Module (Flexible Range Logic) 30
  - 3.3.3 PCI_Target Interface 33
  - 3.3.4 Evaluation Using H.O.T. II PCI_Board 34
- 3.4 Monitor Low-level Software 35
  - 3.4.1 Monitor Driver 37

3.4.2 Monitor Interface 38
3.5 Related Work 41
3.6 Summary 42

4 Tool Middleware Enabling Interoperability 45
4.1 Requirements for an Interoperable Interface 45
4.2 State of the Art 46
4.3 OMIS/OCM: an Interoperable Interface Serving as the Tool Middleware 47
4.4 Extending OCM for DSM Monitoring 48
4.4.1 Specification of an SCI-DSM Extension 48
4.4.2 Specification of a Programming Model Extension 49
4.4.3 Available Services 49
4.4.4 Service Implementation: a Case Study 50
4.5 Interactions between the OCM and the SMiLE Tool Environment 51
4.6 Summary 52

5 A Toolset Supporting Locality Analysis and Optimization 55
5.1 DLV: the Data Layout Visualizer 55
5.1.1 State of the Art 56
5.1.2 Design Guide 60
5.1.3 DLV Display Windows 61
5.1.4 Analyzing a Sample Code 70
5.1.5 Implementation 73
5.2 ARS: An Adaptive Runtime System 79
5.2.1 State of the Art 80
5.2.2 Page Migration Algorithms 82
5.2.3 Graphical User Interface 85
5.2.4 Validation 86
5.2.5 Evaluation of the Migration Algorithms 87
5.3 Integration Using a Global Graphical Environment 92
5.4 Summary 93

6 SIMT: a Multiprocessor Simulator Serving as Evaluation Platform 95
6.1 Architectural Models 95
6.2 Augmint: the Base of SIMT 96
6.2.1 Overview 97
6.2.2 Implementation Detail of Augmint 99
6.3 Adding a New Backend 101
6.3.1 Simulation of Caches 102
6.3.2 Simulation of a Distributed Shared Memory 106
6.3.3 Simulation of the SMiLE Monitor 109
6.3.4 Network Modeling 111
6.3.5 Simulation Structure 112

6.3.6 Macros and Annotations 113
6.4 Using SIMT to Understand Cache Behavior and Remote Access Behavior 113
6.4.1 Simulation Configuration 114
6.4.2 Cache Behavior 114
6.4.3 Capacity and Organization 115
6.4.4 Coherence Protocols 116
6.4.5 Remote Access Behavior 119
6.5 Related Work 120
6.6 Summary 123

7 Performance Evaluation 125
7.1 Benchmark Applications 125
7.2 Memory Access Behavior 128
7.2.1 Performance Sensibility to Remote Access Latency 128
7.2.2 Influence of Data Distribution Policies 129
7.2.3 Data Sharing Pattern of Applications 132
7.2.4 Potential of a Better Data Placement 137
7.3 Locality Optimization 140
7.3.1 Impact of Static Locality Optimization 140
7.3.2 Combining Static Optimization with Dynamic Migration 142
7.3.3 Validating Program Optimization on Real Clusters 144
7.4 Summary 145

8 Conclusions 147
8.1 Summary 148
8.2 Future Work 151
8.2.1 Extension to the Tool Environment 151
8.2.2 Transform to the Real Hardware 153
8.2.3 Enhancing the Monitoring Concept 153
8.3 Concluding Remarks 154

A Software to the SMiLE Hardware Monitor 157
A.1 Functions for Operating the Monitoring Facility 157
A.2 Functions for Event Definition 158
A.3 Functions for Data Handling 159

B Using the Tool Environment 161
B.1 System Requirements 161
B.2 Activating the Global GUI 161

C Using SIMT 163
C.1 Command Line Options 163
C.2 Macros and Annotations 164

Bibliography **167**

Index **183**

List of Figures

2.1 Infrastructure of the tool environment. 14

3.1 The SMiLE SCI node architecture: a PCI-SCI adapter and the hardware monitor card installed in a PC. 23
3.2 Architecture of the SMiLE hardware monitor. 24
3.3 Structure of the counter module. 25
3.4 Coding of the transaction type. 27
3.5 The working principle of the dynamic mode. 28
3.6 B-Link packet at the PCI-SCI card. 29
3.7 The B-Link decoder. 30
3.8 Structure of the range table. 31
3.9 State machine for the dynamic FRL. 32
3.10 Static part of the FRL. 33
3.11 Prototypical setup based on HOT II Board. 35
3.12 Initializing the hardware monitor. 36
3.13 Form of a counter item. 38
3.14 Data transfer from ring buffer to user space. 39
3.15 Structure of the histogram chain. 40

4.1 Global structure: interactions between the OCM and the tool environment. 52

5.1 Overview of DLV display windows. 61
5.2 The main window of DLV. 62
5.3 Actual data transfer demonstration (top: for larger clusters, bottom: for small clusters). 63
5.4 Detailed access numbers (including top left: accesses to pages, bottom left: accesses to memory regions, right: most frequently accesses in descending order). 65
5.5 The *Dominating access* window. 66
5.6 The *Sharing degree* view. 67
5.7 An access diagram on node 1. 68
5.8 The detailed access character of a page (including top left: *Description*, top right: *Section*, bottom left: *Phase*, bottom right: *Read/Write*). 69

5.9 Back to the data structure within the source code. 70
5.10 Memory accesses of some pages on node 1 (SOR). 72
5.11 The detailed access character of Page 65 on node 2 (SOR) (left: *Section*, right: *Read/Write*). 73
5.12 Using the *Frame* component to create a window. 74
5.13 Creating a table. 74
5.14 A sample *Dialog*. 75
5.15 Examples of *Combo box*, *Button*, and *Label*. 75
5.16 Using class *Graphics* to create diagrams. 76
5.17 Data allocation using Round-robin and incoming traffic seen by processor nodes. 84
5.18 ARS GUI display windows (left: *page show*, middle: *runtime migration*, right: *page location*). 86
5.19 Data transfer between ARS, DLV, OCM, and the hardware monitors. 87
5.20 Simulation time using thresholds with different factors. 89
5.21 Migration behavior of WATER (left) and RADIX (right). 89
5.22 Simulation time for different programs using Round-robin. 90
5.23 A graphical environment for program development. 93

6.1 The simulated architecture. 96
6.2 Simulation procedure of Augmint. 98
6.3 Interface between the two simulation components. 99
6.4 Concurrent activities during parallel execution. 100
6.5 Read policy of SMiLE cache protocol (L1: write-through, L2: write-back). 103
6.6 Write policy of the SMiLE cache protocol (L1: write-through, L2: write-back). 104
6.7 Data allocation and stack space for multiple threads. 107
6.8 Examples of data placement on a 4 node system. 108
6.9 Allocation table for the simulation of the SCI-VM. 109
6.10 The SMiLE monitoring concept. 111
6.11 The simulation structure. 112
6.12 Impact of cache size, cache set size, and cache consistency protocols. 115
6.13 Number of cache lines invalidated throughout the execution. 117
6.14 Normalized execution time of various codes and remote latency settings on top of hardware cache coherent (HCC) and non–coherent (NUMA) machines. 118
6.15 Memory access distribution of LU and FFT (top: LU, bottom: FFT). 119
6.16 Comparison of remote memory accesses with different cache sizes (LU). 120

7.1 Simulation time versus remote access latency. 128
7.2 Difference between the data distribution schemes. 130
7.3 Data sharing pattern of FFT (left: read, right: write). 132
7.4 Data sharing pattern of LU (left: read, right: write). 133

7.5 Data sharing pattern of RADIX (left: read, right: write). 133
7.6 Data sharing pattern of WATER (left: read, right: write). 134
7.7 Data sharing pattern of OCEAN (left: read, right: write). 134
7.8 Data sharing pattern of MP3D (left: read, right: write). 135
7.9 Data sharing pattern of SOR (left: read, right: write). 136
7.10 Data sharing pattern of GAUSS (left: read, right: write). 136
7.11 Dominating data distribution of RADIX (top) and FFT (bottom). 138
7.12 Memory accesses of the LU program running on a 4 node system using Round-robin (top: number of accesses, bottom: location of pages). 141
7.13 Comparison of memory accesses before and after optimization (LU program, top: before optimization, bottom: after optimization). 142
7.14 Impact of static data locality optimization. 143
7.15 Simulated speedup on systems with different number of nodes. 144

B.1 The small window for user to input commands. 162

List of Tables

3.1 On-board user space for target FPGAs. 37
3.2 Memory access histogram of a sample program. 41

5.1 Applications and their workload sizes. 88
5.2 Migration behavior (migration: total number of migration, ping-pong: Ping-Pong, multiple: multiple migration, error: incorrect migration). 91

6.1 Simulation parameters. 114

7.1 Description of simulated applications. 125
7.2 Variations in memory access probabilities introduced by different data allocation policies. 139
7.3 Impact of locality optimization on a few codes. 145