

Chapman & Hall/CRC
Computational Science Series

High Performance Computing

Programming and Applications

John Levesque
with Gene Wagenbreth



CRC Press
Taylor & Francis Group

A CHAPMAN & HALL BOOK

Contents

Introduction, xi

CHAPTER 1 ■ Multicore Architectures	1
1.1 MEMORY ARCHITECTURE	1
1.1.1 Why the Memory Wall?	2
1.1.2 Hardware Counters	3
1.1.3 Translation Look-Aside Buffer	4
1.1.4 Caches	6
1.1.4.1 <i>Associativity</i>	6
1.1.4.2 <i>Memory Alignment</i>	11
1.1.5 Memory Prefetching	11
1.2 SSE INSTRUCTIONS	12
1.3 HARDWARE DESCRIBED IN THIS BOOK	14
EXERCISES	16

CHAPTER 2 ■ The MPP: A Combination of Hardware and Software	19
2.1 TOPOLOGY OF THE INTERCONNECT	20
2.1.1 Job Placement on the Topology	21
2.2 INTERCONNECT CHARACTERISTICS	22
2.2.1 The Time to Transfer a Message	22
2.2.2 Perturbations Caused by Software	23
2.3 NETWORK INTERFACE COMPUTER	24
2.4 MEMORY MANAGEMENT FOR MESSAGES	24

2.5	HOW MULTICORES IMPACT THE PERFORMANCE OF THE INTERCONNECT	25
	EXERCISES	25
CHAPTER 3 ■ How Compilers Optimize Programs		27
3.1	MEMORY ALLOCATION	27
3.2	MEMORY ALIGNMENT	28
3.3	VECTORIZATION	29
3.3.1	Dependency Analysis	31
3.3.2	Vectorization of IF Statements	32
3.3.3	Vectorization of Indirect Addressing and Strides	34
3.3.4	Nested DO Loops	35
3.4	PREFETCHING OPERANDS	36
3.5	LOOP UNROLLING	37
3.6	INTERPROCEDURAL ANALYSIS	38
3.7	COMPILER SWITCHES	38
3.8	FORTRAN 2003 AND ITS INEFFICIENCIES	39
3.8.1	Array Syntax	40
3.8.2	Using Optimized Libraries	42
3.8.3	Passing Array Sections	42
3.8.4	Using Modules for Local Variables	43
3.8.5	Derived Types	43
3.9	SCALAR OPTIMIZATIONS PERFORMED BY THE COMPILER	44
3.9.1	Strength Reduction	44
3.9.2	Avoiding Floating Point Exponents	46
3.9.3	Common Subexpression Elimination	47
	EXERCISES	48
CHAPTER 4 ■ Parallel Programming Paradigms		51
4.1	HOW CORES COMMUNICATE WITH EACH OTHER	51
4.1.1	Using MPI across All the Cores	51

4.1.2	Decomposition	52
4.1.3	Scaling an Application	53
4.2	MESSAGE PASSING INTERFACE	55
4.2.1	Message Passing Statistics	55
4.2.2	Collectives	56
4.2.3	Point-to-Point Communication	57
	4.2.3.1 <i>Combining Messages into Larger Messages</i>	58
	4.2.3.2 <i>Preposting Receives</i>	58
4.2.4	Environment Variables	61
4.2.5	Using Runtime Statistics to Aid MPI-Task Placement	61
4.3	USING OPENMP™	62
4.3.1	Overhead of Using OpenMP™	63
4.3.2	Variable Scoping	64
4.3.3	Work Sharing	66
4.3.4	False Sharing in OpenMP™	68
4.3.5	Some Advantages of Hybrid Programming: MPI with OpenMP™	70
	4.3.5.1 <i>Scaling of Collectives</i>	70
	4.3.5.2 <i>Scaling Memory Bandwidth Limited MPI Applications</i>	70
4.4	POSIX® THREADS	71
4.5	PARTITIONED GLOBAL ADDRESS SPACE LANGUAGES (PGAS)	77
4.5.1	PGAS for Adaptive Mesh Refinement	78
4.5.2	PGAS for Overlapping Computation and Communication	78
4.5.3	Using CAF to Perform Collective Operations	79
4.6	COMPILERS FOR PGAS LANGUAGES	83
4.7	ROLE OF THE INTERCONNECT	85
	EXERCISES	85

CHAPTER 5 ■ A Strategy for Porting an Application to a Large MPP System	87
5.1 GATHERING STATISTICS FOR A LARGE PARALLEL PROGRAM	89
EXERCISES	96
CHAPTER 6 ■ Single Core Optimization	99
6.1 MEMORY ACCESSING	99
6.1.1 Computational Intensity	102
6.1.2 Looking at Poor Memory Utilization	103
6.2 VECTORIZATION	109
6.2.1 Memory Accesses in Vectorized Code	110
6.2.2 Data Dependence	112
6.2.3 IF Statements	119
6.2.4 Subroutine and Function Calls	129
6.2.4.1 <i>Calling Libraries</i>	135
6.2.5 Multinested DO Loops	136
6.2.5.1 <i>More Efficient Memory Utilization</i>	145
6.3 SUMMARY	156
6.3.1 Loop Reordering	156
6.3.2 Index Reordering	157
6.3.3 Loop Unrolling	157
6.3.4 Loop Splitting (Loop Fission)	157
6.3.5 Scalar Promotion	157
6.3.6 Removal of Loop-Independent IFs	158
6.3.7 Use of Intrinsics to Remove IFs	158
6.3.8 Strip Mining	158
6.3.9 Subroutine Inlining	158
6.3.10 Pulling Loops into Subroutines	158
6.3.11 Cache Blocking	159
6.3.12 Loop Jamming (Loop Fusion)	159
EXERCISES	159

CHAPTER 7 ■ Parallelism across the Nodes	161
7.1 LESLIE3D	162
7.2 PARALLEL OCEAN PROGRAM (POP)	164
7.3 SWIM	169
7.4 S3D	171
7.5 LOAD IMBALANCE	174
7.5.1 SWEEP3D	177
7.6 COMMUNICATION BOTTLENECKS	179
7.6.1 Collectives	179
7.6.1.1 <i>Writing One's Own Collectives to Achieve Overlap with Computation</i>	179
7.6.2 Point to Point	180
7.6.3 Using the "Hybrid Approach" Efficiently	180
7.6.3.1 <i>SWIM Benchmark</i>	181
7.7 OPTIMIZATION OF INPUT AND OUTPUT (I/O)	182
7.7.1 Parallel File Systems	183
7.7.2 An Inefficient Way of Doing I/O on a Large Parallel System	184
7.7.3 An Efficient Way of Doing I/O on a Large Parallel System	184
EXERCISES	185
CHAPTER 8 ■ Node Performance	187
8.1 WUPPERTAL WILSON FERMION SOLVER: WUPWISE	187
8.2 SWIM	189
8.3 MGRID	192
8.4 APPLU	192
8.5 GALGEL	194
8.6 APSI	196
8.7 EQUAKE	199
8.8 FMA-3D	201
8.9 ART	203

x ■ Contents

8.10 ANOTHER MOLECULAR MECHANICS PROGRAM (AMMP)	204
8.11 SUMMARY	206
EXERCISES	207
CHAPTER 9 ■ Accelerators and Conclusion	209
9.1 ACCELERATORS	210
9.1.1 Using Extensions to OpenMP™ Directives for Accelerators	211
9.1.2 Efficiency Concerns When Using an Accelerator	212
9.1.3 Summary for Using Accelerators	213
9.1.4 Programming Approaches for Using Multicore Nodes with Accelerators	214
9.2 CONCLUSION	214
EXERCISES	215
APPENDIX A: COMMON COMPILER DIRECTIVES, 217	
APPENDIX B: SAMPLE MPI ENVIRONMENT VARIABLES, 219	
REFERENCES, 221	
INDEX, 223	