

Performance Optimization of Numerically Intensive Codes

Stefan Goedecker

Commissariat à l'énergie atomique
Grenoble, France

Adolfy Hoisie

Los Alamos National Laboratory
Los Alamos, New Mexico

siam[®]

Society for Industrial and Applied Mathematics
Philadelphia

Contents

Preface	ix
1 Introduction	1
2 Notions of Computer Architecture	3
2.1 The on-chip parallelism of superscalar architectures	3
2.2 Overview of the memory hierarchy of RISC architectures	7
2.3 Mapping rules for caches	10
2.4 A taxonomy of cache misses	11
2.5 TLB misses	12
2.6 Multilevel cache configurations	12
2.7 Characteristics of memory hierarchies on some common machines . .	13
2.8 Parallel architectures	13
2.8.1 Shared memory architectures	13
2.8.2 Distributed memory architectures	16
2.8.3 Distributed shared memory architectures	22
2.9 A comparison between vector and superscalar architectures	23
3 A Few Basic Efficiency Guidelines	27
3.1 Selection of best algorithm	27
3.2 Use of efficient libraries	27
3.3 Optimal data layout	29
3.4 Use of compiler optimizations	30
3.5 Basic optimizations done by the compiler	30
4 Timing and Profiling of a Program	37
4.1 Subroutine-level profiling	38
4.2 Tick-based profiling	39
4.3 Timing small sections of your program	40
4.4 Assembler output	42
4.5 Hardware performance monitors	42
4.6 Profiling parallel programs	44

5	Optimization of Floating Point Operations	47
5.1	Fused multiply-add instructions	48
5.2	Exposing instruction-level parallelism in a program	48
5.3	Software pipelining	50
5.4	Improving the ratio of floating point operations to memory accesses	53
5.5	Running out of registers	55
5.6	An example where automatic loop unrolling fails	59
5.7	Loop unrolling overheads	60
5.8	Aliasing	60
5.9	Array arithmetic in Fortran90	64
5.10	Operator overloading in C++	67
5.11	Elimination of floating point exceptions	67
5.12	Type conversions	67
5.13	Sign conversions	68
5.14	Complex arithmetic	69
5.15	Special functions	69
5.16	Eliminating overheads	72
	5.16.1 If statements	72
	5.16.2 Loop overheads	76
	5.16.3 Subroutine calling overheads	77
5.17	Copy overheads in Fortran90	78
5.18	Attaining peak speed	78
6	Optimization of Memory Access	79
6.1	An illustration of the memory access times on RISC machines	79
6.2	Performance of various computers for unit and large stride data access	82
6.3	Loop reordering for optimal data locality	85
6.4	Loop fusion to reduce unnecessary memory references	87
6.5	Data locality and the conceptual layout of a program	88
6.6	Cache thrashing	89
6.7	Experimental determination of cache and TLB parameters	91
6.8	Finding optimal strides	93
6.9	Square blocking	95
6.10	Line blocking	98
6.11	Prefetching	103
6.12	Misalignment of data	105
7	Miscellaneous Optimizations	107
7.1	Balancing the load of the functional units	107
7.2	Accessing the instructions	107
7.3	I/O: Writing to and reading from files	108
7.4	Memory fragmentation in Fortran90	108
7.5	Optimizations for vector architectures	110

8	Optimization of Parallel Programs	113
8.1	Ideal and observed speedup	113
8.2	Message passing libraries	115
8.3	Data locality	116
8.4	Load balancing	117
8.5	Minimizing the surface-to-volume ratio in grid-based methods	117
8.6	Coarse-grain parallelism against fine-grain parallelism	118
8.7	Adapting parallel programs to the computer topology	120
9	Case Studies	121
9.1	Matrix-vector multiplication	121
9.2	Sparse matrix-vector multiplication	124
9.3	Two loops from a configuration interaction program	126
9.4	A two-dimensional wavelet transform	131
9.5	A three-dimensional fast Fourier transform	135
9.5.1	Vector machines	136
9.5.2	RISC machines	137
9.5.3	Parallel machines	138
9.6	Multigrid methods on parallel machines	140
9.7	A real-world electronic structure code	145
10	Benchmarks	147
	Appendix	151
A.1	Timing routine for BLAS library	151
A.2	MPI timing routine	152
A.3	Program that should run at peak speed	155
A.4	Program for memory testing	159
A.5	Program to test suitability of parallel computers for fine-grained tasks	163
A.6	Unrolled CI loop structure	164
	Bibliography	167
	Index	169