

---

# Contents

<b>Foreword</b>	<b>xiii</b>
<b>Preface</b>	<b>xv</b>
<b>About the authors</b>	<b>xxi</b>
<b>List of acronyms and abbreviations</b>	<b>xxiii</b>
<b>1 Modern processors</b>	<b>1</b>
1.1 Stored-program computer architecture . . . . .	1
1.2 General-purpose cache-based microprocessor architecture . . . . .	2
1.2.1 Performance metrics and benchmarks . . . . .	3
1.2.2 Transistors galore: Moore's Law . . . . .	7
1.2.3 Pipelining . . . . .	9
1.2.4 Superscalarity . . . . .	13
1.2.5 SIMD . . . . .	14
1.3 Memory hierarchies . . . . .	15
1.3.1 Cache . . . . .	15
1.3.2 Cache mapping . . . . .	18
1.3.3 Prefetch . . . . .	20
1.4 Multicore processors . . . . .	23
1.5 Multithreaded processors . . . . .	26
1.6 Vector processors . . . . .	28
1.6.1 Design principles . . . . .	29
1.6.2 Maximum performance estimates . . . . .	31
1.6.3 Programming for vector architectures . . . . .	32
<b>2 Basic optimization techniques for serial code</b>	<b>37</b>
2.1 Scalar profiling . . . . .	37
2.1.1 Function- and line-based runtime profiling . . . . .	38
2.1.2 Hardware performance counters . . . . .	41
2.1.3 Manual instrumentation . . . . .	45
2.2 Common sense optimizations . . . . .	45
2.2.1 Do less work! . . . . .	45
2.2.2 Avoid expensive operations! . . . . .	46
2.2.3 Shrink the working set! . . . . .	47

2.3	Simple measures, large impact . . . . .	47
2.3.1	Elimination of common subexpressions . . . . .	47
2.3.2	Avoiding branches . . . . .	48
2.3.3	Using SIMD instruction sets . . . . .	49
2.4	The role of compilers . . . . .	51
2.4.1	General optimization options . . . . .	52
2.4.2	Inlining . . . . .	52
2.4.3	Aliasing . . . . .	53
2.4.4	Computational accuracy . . . . .	54
2.4.5	Register optimizations . . . . .	55
2.4.6	Using compiler logs . . . . .	55
2.5	C++ optimizations . . . . .	56
2.5.1	Temporaries . . . . .	56
2.5.2	Dynamic memory management . . . . .	59
2.5.3	Loop kernels and iterators . . . . .	60
<b>3</b>	<b>Data access optimization</b>	<b>63</b>
3.1	Balance analysis and lightspeed estimates . . . . .	63
3.1.1	Bandwidth-based performance modeling . . . . .	63
3.1.2	The STREAM benchmarks . . . . .	67
3.2	Storage order . . . . .	69
3.3	Case study: The Jacobi algorithm . . . . .	71
3.4	Case study: Dense matrix transpose . . . . .	74
3.5	Algorithm classification and access optimizations . . . . .	79
3.5.1	$O(N)/O(N)$ . . . . .	79
3.5.2	$O(N^2)/O(N^2)$ . . . . .	79
3.5.3	$O(N^3)/O(N^2)$ . . . . .	84
3.6	Case study: Sparse matrix-vector multiply . . . . .	86
3.6.1	Sparse matrix storage schemes . . . . .	86
3.6.2	Optimizing JDS sparse MVM . . . . .	89
<b>4</b>	<b>Parallel computers</b>	<b>95</b>
4.1	Taxonomy of parallel computing paradigms . . . . .	96
4.2	Shared-memory computers . . . . .	97
4.2.1	Cache coherence . . . . .	97
4.2.2	UMA . . . . .	99
4.2.3	ccNUMA . . . . .	100
4.3	Distributed-memory computers . . . . .	102
4.4	Hierarchical (hybrid) systems . . . . .	103
4.5	Networks . . . . .	104
4.5.1	Basic performance characteristics of networks . . . . .	104
4.5.2	Buses . . . . .	109
4.5.3	Switched and fat-tree networks . . . . .	110
4.5.4	Mesh networks . . . . .	112
4.5.5	Hybrids . . . . .	113

<b>5</b>	<b>Basics of parallelization</b>	<b>115</b>
5.1	Why parallelize? . . . . .	115
5.2	Parallelism . . . . .	116
5.2.1	Data parallelism . . . . .	116
5.2.2	Functional parallelism . . . . .	119
5.3	Parallel scalability . . . . .	120
5.3.1	Factors that limit parallel execution . . . . .	120
5.3.2	Scalability metrics . . . . .	122
5.3.3	Simple scalability laws . . . . .	123
5.3.4	Parallel efficiency . . . . .	125
5.3.5	Serial performance versus strong scalability . . . . .	126
5.3.6	Refined performance models . . . . .	128
5.3.7	Choosing the right scaling baseline . . . . .	130
5.3.8	Case study: Can slower processors compute faster? . . . . .	131
5.3.9	Load imbalance . . . . .	137
<b>6</b>	<b>Shared-memory parallel programming with OpenMP</b>	<b>143</b>
6.1	Short introduction to OpenMP . . . . .	143
6.1.1	Parallel execution . . . . .	144
6.1.2	Data scoping . . . . .	146
6.1.3	OpenMP worksharing for loops . . . . .	147
6.1.4	Synchronization . . . . .	149
6.1.5	Reductions . . . . .	150
6.1.6	Loop scheduling . . . . .	151
6.1.7	Tasking . . . . .	153
6.1.8	Miscellaneous . . . . .	154
6.2	Case study: OpenMP-parallel Jacobi algorithm . . . . .	156
6.3	Advanced OpenMP: Wavefront parallelization . . . . .	158
<b>7</b>	<b>Efficient OpenMP programming</b>	<b>165</b>
7.1	Profiling OpenMP programs . . . . .	165
7.2	Performance pitfalls . . . . .	166
7.2.1	Ameliorating the impact of OpenMP worksharing constructs . . . . .	168
7.2.2	Determining OpenMP overhead for short loops . . . . .	175
7.2.3	Serialization . . . . .	177
7.2.4	False sharing . . . . .	179
7.3	Case study: Parallel sparse matrix-vector multiply . . . . .	181
<b>8</b>	<b>Locality optimizations on ccNUMA architectures</b>	<b>185</b>
8.1	Locality of access on ccNUMA . . . . .	185
8.1.1	Page placement by first touch . . . . .	186
8.1.2	Access locality by other means . . . . .	190
8.2	Case study: ccNUMA optimization of sparse MVM . . . . .	190
8.3	Placement pitfalls . . . . .	192
8.3.1	NUMA-unfriendly OpenMP scheduling . . . . .	192

8.3.2	File system cache . . . . .	194
8.4	ccNUMA issues with C++ . . . . .	197
8.4.1	Arrays of objects . . . . .	197
8.4.2	Standard Template Library . . . . .	199
<b>9</b>	<b>Distributed-memory parallel programming with MPI</b>	<b>203</b>
9.1	Message passing . . . . .	203
9.2	A short introduction to MPI . . . . .	205
9.2.1	A simple example . . . . .	205
9.2.2	Messages and point-to-point communication . . . . .	207
9.2.3	Collective communication . . . . .	213
9.2.4	Nonblocking point-to-point communication . . . . .	216
9.2.5	Virtual topologies . . . . .	220
9.3	Example: MPI parallelization of a Jacobi solver . . . . .	224
9.3.1	MPI implementation . . . . .	224
9.3.2	Performance properties . . . . .	230
<b>10</b>	<b>Efficient MPI programming</b>	<b>235</b>
10.1	MPI performance tools . . . . .	235
10.2	Communication parameters . . . . .	239
10.3	Synchronization, serialization, contention . . . . .	240
10.3.1	Implicit serialization and synchronization . . . . .	240
10.3.2	Contention . . . . .	243
10.4	Reducing communication overhead . . . . .	244
10.4.1	Optimal domain decomposition . . . . .	244
10.4.2	Aggregating messages . . . . .	248
10.4.3	Nonblocking vs. asynchronous communication . . . . .	250
10.4.4	Collective communication . . . . .	253
10.5	Understanding intranode point-to-point communication . . . . .	253
<b>11</b>	<b>Hybrid parallelization with MPI and OpenMP</b>	<b>263</b>
11.1	Basic MPI/OpenMP programming models . . . . .	264
11.1.1	Vector mode implementation . . . . .	264
11.1.2	Task mode implementation . . . . .	265
11.1.3	Case study: Hybrid Jacobi solver . . . . .	267
11.2	MPI taxonomy of thread interoperability . . . . .	268
11.3	Hybrid decomposition and mapping . . . . .	270
11.4	Potential benefits and drawbacks of hybrid programming . . . . .	273
<b>A</b>	<b>Topology and affinity in multicore environments</b>	<b>277</b>
A.1	Topology . . . . .	279
A.2	Thread and process placement . . . . .	280
A.2.1	External affinity control . . . . .	280
A.2.2	Affinity under program control . . . . .	283
A.3	Page placement beyond first touch . . . . .	284

**B Solutions to the problems**

**287**

**Bibliography**

**309**

**Index**

**323**