

Index

Note: Page numbers followed by *f* indicate figures and *t* indicate tables.

A

- ABarrier class, 203, 419–420
 - algorithm, 205, 205*f*
 - implementation, 204–206
- Accelerators, offloading code blocks to, 292–297
- Acquire operations, 161
- Acquire semantics, 179
- Acquire-release fence, 161
- Acquire-release memory model, 175
 - sequential consistency compared with, 177–178
- Advanced reads, 157–158
- Affinity identity, 470
- Algorithms
 - ABarrier class, 205, 205*f*
 - Box-Muller, 97–98
 - divide and conquer, 222
 - image-filtering, 401
 - lock-free, 125, 171–174
 - as alternative to critical sections, 171
 - AReduction class, 172–174
 - as optimistic approach, 171
 - updating global data item, 171–172
 - parallel quicksort, 282–285
 - parallel_do, 310, 328–329
 - parallel_for, 309, 385
 - example of usage, 317–320
 - parallel_for_each, 310, 327–328
 - parallel_invoke, 310, 329–330
 - parallel_reduce, 309, 320–327, 418
 - domain decomposition strategy in, 421
 - parallel search for minimum value, 324–327
 - recursive area computation, 321–324
 - parallel_scan, 309–310
 - parallel_sort, 310
 - pipeline, 310
 - recursive, 242
 - area under curve, 277–282, 280*f*, 321–324
 - divide and conquer, 222
 - OpenMP versions, 277–282
 - parallel search for minimum value, 324–327
 - quicksort, 282–285
 - TBB, 309–310
 - built-in range classes, 314–315
 - fork-join parallel pattern, 315–317
 - integer range example, 313–314
 - operation of, 312–317
 - operation of high-level, 312–317
 - parallel_for example, 317–320
 - parallel_for operation, 319–320
 - task scheduler, 469
 - thread pool scheduling, 461–462
- Aligned memory allocation, 395–396
- Area computation
 - with fixed number of tasks, 332–334
 - non-blocking style, 336–337
 - tasks wait for children, 334–336
- Area under curve, 77–78
 - blocking style, 277–280, 278*f*
 - nonblocking style, 280–282, 280*f*
 - NPool, 364–365
 - TBB parallel_reduce, 321–324
 - public data items, 322
 - reduction in, 322–324
 - split constructor, 322
- AReduction class, 172–174
- Assembly line, 423
- Associative containers, 125, 126
- Atomic classes, 171
- Atomic data types, 121, 167
- atomic directive, 121, 122
- Atomic operations, 121–124, 167
 - as alternative to mutex locking, 121, 126
 - C++11, 167, 169–170
 - OpenMP, 101, 122–124
 - restricted scope, 126
 - TBB, 167
 - Windows threads, 167
- Atomic pointers, 172–173
- Atomic synchronizations, 180–185
 - circular buffers, 182–185
 - SpinLock class, 180–185
 - tbb::atomic<T>, 185–187
 - Windows, 187–190
- Atomic variables, 102, 161
- Automatic initialization, TBB, 311–312
- Automatic parallelization, 225–226
 - controlling in OpenMP, 233
 - OpenMP work-sharing directive for, 239
- Automatic work-sharing, 392

B

Bandwidth, 3
 Barrier class, 203, 385
 barrier directive, 243
 behavior of, 252–254
 Pthreads and vath barrier functions compared with, 252
 Barrier synchronization, 142–144, 273–274, 385
 ABarrier implementation, 146–147
 blocking barriers, 206–210, 207*f*
 examples, 207–210
 OpenMP, 243–244, 389
 public interface, 204
 spin, 203–206
 SPool molecular dynamics implementation, 390–391
 SynchP compared with, 201
 tasks and, 257, 260, 260*f*
 threads and, 257
 vath library classes for, 203–204
 public interface, 204
 using, 204
 Barriers, 193
 ABarrier class, 203, 419–420
 algorithm, 205, 205*f*
 implementation, 204–206
 idle-wait *versus* spin-wait, 390–391
 SpBarrier class, 203, 385
 among tasks, 274–277
 TBarrier class, 203
 _beginthread(), 41–42, 43
 _beginthreadex(), 41–42, 43
 BLock class, 195, 438
 examples, 197–198
 interfaces, 196
 spin wait implementation, 198–199
 using, 196–197
 Blocked state, 26
 Blocking barriers, 193, 206–210, 207*f*
 examples, 207–210
 SPMD thread pool, 208–210
 main() driving worker activity, 209–210
 worker thread launching, 208–209
 worker thread termination, 210
 using, 206–207
 Boolean locks, 193, 195–199
 in handshake synchronization,
 432, 433*f*
 implementing spin waits, 198–199
 pipelines and, 438
 Boost
 scoped locking method, 116–117
 SpinLock class, 180–182
 Boost, 50

Box-Muller algorithm, 97–98
 Brownian motion, 377–378
 Buffers
 circular, 182–185
 synchronization, 183–185
 RingBuffer class, 182, 183–184, 436
 stack, 18
 working, circular array of, 431–432
 write, 9, 157–158
 Busy wait, 104, 129, 159. *See also* Spin waits

C

C, 225
 preprocessor, 390, 395–396
 C++, 225, 491
 callable objects, 495
 function objects, 495
 syntax, 495–500
 Lambda expressions, 495
 C Runtime Library (CRT), 41, 43
 C++ Standard Template Library, containers, 124
 C++11
 atomic data types, 167
 atomic operations, 167, 169–170
 CAS operation, 171
 condition variables, 137–139
 interface for, 137
 futures and promises, 147–152
 std::future<T> class, 147–148
 std::promise<T> class, 150–152
 idle waits, 137, 138–139
 implementations, 50
 lambda expressions introduced in, 500–501
 memory models, 175–176
 Reduction<T>, 112–113
 std::atomic<T> class, 167–168
 Monte Carlo computation of π with, 169–170
 thread attributes, 31
 thread data types, 30
 thread function returns, 34
 thread management interfaces, 29
 thread_local keyword, 90–91
 time durations, 81
 timed wait, 137, 140–142
 C++11 thread library, 49–63
 condition variables, 131, 132
 creating and running threads, 51–54
 data types, 50–51
 error reporting, 50
 headers, 49–50
 mutexes, 116–118
 scoped lock class templates, 116–118

- scope of thread objects, 54
 - stack size, 53
 - thread management, 55
 - launching worker team, 61–63
 - passing values by reference, 58–59
 - transferring thread ownership, 60
 - worker threads, 55
 - threads as objects, 52–53
 - Cache coherency issue, 9, 154–155, 154*f*
 - false sharing problem, 155
 - memory consistency issue and, 155
 - SMP platforms, 154, 154*f*
 - Cache lines, 8
 - Cache memory, 8
 - issues, 9
 - performance comparison and, 393
 - Callbacks, 312
 - implementing, 497
 - Cancellation of parallel constructs, 287–292
 - cancel directive, 287–288
 - checking for requests, 288–289
 - parallel regions, 219, 289–290
 - taskgroup regions, 290–292
 - cancellation point directive, 288–289
 - CancelTeam() utility, 67–68, 78–79
 - capture clause, 122
 - CAS. *See* Compare and Swap
 - Central processing unit (CPU), 1, 2, 2*f*
 - limits in multiprocessor systems, 3
 - load-store architectures, 8
 - performance, 2–3
 - registers, 11
 - Check and act operations, 127
 - Child stealing, 317, 467
 - Cilk Plus, 307–308, 316, 459–460
 - continuation stealing, 317, 467
 - Circular array of working buffers, 431–432
 - Circular buffers, 182–185
 - synchronization, 183–185
 - Clauses, 230
 - capture, 122
 - data-sharing attributes, 234, 240–242, 240*t*
 - declare simd directive, 304
 - depend, 259, 260, 274, 355
 - functional, 234
 - functional parallel, 235
 - map, 293
 - read, 122
 - seq_cst, 122
 - simd directive, 302–303
 - task directive and, 258–260
 - types of, 234
 - update, 122
 - write, 122
 - CloseHandle(), 43
 - Cluster computing, 6, 7*f*
 - collapse(n) directive, 239, 386
 - loop fusion and, 386
 - microtasking and, 407–408
 - Compare and Swap (CAS), 171–172
 - Compiler optimizations
 - conflicts with sequential, 158–159
 - memory accesses and, 157–158
 - Compiling and running examples, 493
 - Complex Fourier transforms, 426–427
 - Complex matrices, 426–427
 - Computing platforms, 2–7, 2*f*
 - Conceptual variables, 229–230
 - Concurrency, 1
 - parallel processing and, 20
 - pipeline pattern for, 423–424, 424*f*
 - Concurrent containers, 125
 - TBB classes for, 125–126
 - Concurrent execution, 25
 - Concurrent programming
 - benefits of, 26–28
 - GUIs and, 28
 - I/O and, 27
 - Concurrent queues, pipelines and, 438
 - Condition variables, 130
 - C++11, 137–139
 - interface for, 137
 - thread library, 131, 132
 - in idle waits, 131–132
 - Pthreads
 - interface for, 132
 - wait protocol, 133–134
 - waiting on, 133
 - Windows API, 135
 - Windows threads, 131, 132, 135–136
 - interface for, 135
- CONDITION_VARIABLE, 131
- Constant power dissipation, 5
- Consume semantics, 179
- Consumer threads, 212–213, 214, 215, 436
- Container thread safety, 124–126
- Contiguous data, 428
- Continuation passing, 467–468, 470–474, 475
- Continuation stealing, 317, 467
- Continuation task, 315, 316–317, 466, 467*f*, 474
- Control parallelism, 375, 414, 423
- Coprocessors, 12, 14
- Copy constructor, 322
- Copy semantics, 58

Core registers, 8
 Correlated Gaussian fluctuations, 377–378, 380–381
 hybrid MPI-threads example, 369–373
 running parallel routines in parallel, 366
 client code, 367–369
 main parallel subroutine, 366–367
 Correlated Gaussian random numbers, 377
 Correlated Gaussian vectors, 377
 CPP11_ENV environment variable, 493
 CPU. *See* Central processing unit
 CPU registers, 11
 CpuTimer, 69
 CreateThread(), 41
 critical directive, 243, 244–245
 Critical sections, 103
 exception safe, 117
 lock free algorithms as alternative to, 171
 with mutex and lock_guard object, 117
 OpenMP, 244–245
 mutex locking *versus* named, 245
 naming, 245
 TBB interface for, 120
 CRITICAL_SECTION interface, 114
 CRT. *See* C Runtime Library
 CUDA, 13

D

Data alignment, 398–399
 Data dependencies, 303, 401, 414
 control parallelism and, 414
 generic approaches to handling, 420–422
 pipelining and, 440
 separating hyperplane for handling, 414
 in stencil codes, 414
 white-black ordering for handling, 414–415
 Data operations, 160
 Data parallelism, 375, 392, 423
 general program structure, 384–385, 384*f*
 vectorization and, 385
 Data types
 atomic, 121, 167
 C++11 thread library, 50–51
 Pthreads, 33–34
 thread representation with, 30
 Windows threads, 41–43
 Data-sharing attributes clauses, 234, 240–242, 240*t*
 Deadlocks, 117
 avoiding in NPool operation, 352
 dynamic tasks and, 352
 worker thread and, 352, 353
 declare simd directive, 303–304
 clauses, 304

Delayed writes, 157–158
 Dennard, Robert, 5
 Dennard scaling, 5, 6, 6*t*
 depend clause, 259, 260, 274, 355
 Dependence type, 274
 Deques, 460, 461
 Device scope ICVs, 230
 Diagonal swap, 420–422, 421*f*
 Directed acyclic graph, 267*f*
 cell and graph classes, 268–269
 constructing, 269–270, 269*f*
 traversing, 267–272
 NPool, 361–364
 parallel context, 267–270
 updating cells, 268
 Directives, 74, 225
 adding to parallel constructs, 246–248
 parallel, 227
 task, 258–260
 Dispatch(), 66, 68
 distribute construct, 294–297
 Distributed memory multiprocessor systems, 4–5, 4*f*
 message overhead in, 380
 molecular dynamics problem and, 380
 Divide and conquer algorithms, 222
 DMonitor class, 381–382
 Domain decomposition strategy, 77, 421
 Double-ended queue, 351
 Dynamic task groups, 349
 Dynamic tasks, 351
 deadlocks and, 352
 Dynamic thread pools, 227–228, 232–233, 342

E

Empty tasks, 480–481
 task recycling and, 486
 Encapsulation
 complexity control with, 127
 mutexes and, 127
 shared variables, 111–112
 Environment variables, 225
 CPP11_ENV, 493
 OMP_CANCELLATION, 288
 OMP_PLACES, 298, 299
 OMP_PROC_BIND, 298
 Equations of motion, integration of, 378–380
 parallel algorithm, 379–380
 Errno, 32
 Error checking, conventions for, 32–33
 Error handling, 31–32
 Pthreads error reporting, 32–33

- Error reporting
 - C++11 thread library, 50
 - Pthreads, 32–33
 - Unix and C conventions, 32
- Event predicates, 131
- Event synchronizations, 35–36, 101, 102, 129
 - data elements in, 131
 - declaring objects for, 131–132
 - explicit tasks and, 257–258
 - task-centric regions and, 258
- Event-like synchronizations, 459
- Events, 129
- Exceptions, mutexes and, 117
- Execution models, programming models and, 25–26
- Explicit tasks, 229
 - event synchronizations and, 257–258
 - Monte Carlo computation of π with, 286–287
 - task directive creating, 258–259
- Explicit thread pools, 342, 342f
- External computational devices, 12–14, 12f
 - offloading code blocks to, OpenMP, 292–297

F

- Fair mutexes, 104
- False sharing problem, 155, 393
- Fast Fourier Transform (FFT), 425, 453–454
 - pipelined versions, 431–433
 - circular array of working buffers, 431–432
 - with handshake synchronization, 432–435
 - with producer-consumer synchronization, 435–436, 437–438
 - routines for, 427–428, 455–457
 - half of two-dimensional, 456–457
 - one-dimensional, 427–428, 455–456
 - real-valued functions, 457
 - two-dimensional, 428, 456
 - sequential, 426–427, 426f, 427f, 428–430
- FFT. *See* Fast Fourier Transform
- filter interface class, 445
- Finite-difference discretization, 402, 452–453
- Flat MPI distributed memory programming, 7
- flush directive, 165–166
- for loops, parallel, 248–249
- Fork-join pattern, 63–64, 222, 227
 - parallel_for implementing, 317, 319
 - parallel search for minimum value, 326, 327
 - in recursive area computation, 322, 323f
 - TBB use of, 315–317
 - task spawning paradigms, 466–467
- FORTTRAN, 225

- Fourier coefficients, 426, 428, 454
- Fourier transforms, 453–454
- Function object class, 319
- Function objects, 51, 52, 319, 495
 - abilities of, 497–498
 - generic programing, 497–498
 - handling functions with internal states, 497
 - runtime performance, 497
 - callback implementation with, 497
 - examples, 498–500
 - lambda function syntax for, 331
 - syntax, 495–500
 - TBB, 500
- Function pointers, 496
- Functional clauses, 234
- Functional parallel clauses, 235
- Functions
 - library, 23
 - error handling and, 31–32
 - reentrant versions, 85
 - thread safety of, 84–85
 - persistent state, 86–87
- Functors. *See* Function objects
- Funneled multithreading support, 370
- Futures, 130, 147–152
 - I/O operations and, 149
 - std::future<T> class, 147–148

G

- Gaussian random generator, 96–98
 - Box-Muller algorithm, 97–98
 - examples, 98
- Gaussian random numbers, 377
- Gauss-Seidel method, 403, 415f, 440
 - over-relaxed, 403
- Gauss-Seidel successive over-relaxation method, 414–417
 - diagonal swap version, 421
 - parallel implementations, 417–419
 - OpenMP and NPool macrotasking, 417–418
 - OpenMP microtasking, 417
 - TBB microtasking, 418–419
 - performance issues, 419–420
 - pipelined version, 440–444, 441f
 - sequential implementation, 415–417
 - with white-black ordering, 414–415
- GCC, 50, 491
- Generic programming, 194, 199, 497–498
- get_future(), 150
- Global data items, lock-free updating, 171–172
- Global order in sequential consistency, 177, 177f

Global pointers, 22, 54, 56
 SPool objects, 66–67
 Global scope ICVs, 230
 Global task queue, 460
 Global variables, 17, 21
 error reporting and, 32
 shared variables compared with, 241–242
 GNU 4.8, 50
 GNU 4.9, 292
 GPU. *See* Graphical Processing Unit
 Graphical interfaces (GUIs), concurrent programming
 benefits and, 28
 Graphical Processing Unit (GPU), 12–14, 292
 GUIs. *See* Graphical interfaces

H

Half of two-dimensional FFT routines, 456–457
 HANDLE, 41
 Handshake synchronization, 432–433, 433*f*
 examples, 433–435
 OpenMP implementation of, 433–435
 “Happens before” relationships, 174–175
 Hardware threads, 10
 Harmonic oscillators, 376
 Heap, 17
 Heat diffusion coefficient, 425
 Heterogeneous computing, 13
 Hierarchical memory system, 8, 8*f*, 158*f*
 performance comparison and, 393
 Hierarchical relations among task, 464–465
 High-level synchronization tools, 191–193
 Hybrid MPI-Threads programming model, 7
 correlated Gaussian fluctuations with, 369–373
 Hyperthreading, 10, 24, 298–299, 391–392, 484–485

I

IBM BlueGene, 5, 301
 IBM BlueGreen, 10
 IBM Power 8, 365–366
 IBM Power6 architecture, 160
 ICVs. *See* Internal control variables
 Idle barrier synchronization, 203–206
 Idle waits, 104, 129–130, 203, 385
 barrier synchronization, 142–144
 barriers, 390–391
 C++11, 137, 138–139
 on condition, 195
 condition variables in, 131–132
 examples, 139–142

Pthreads, 132–135
 atomic mutex lock and wait in, 134–135
 predicate check on return, 134
 wait protocol, 133–134
 waiting on condition variables, 133
 waking up idle threads, 135
 spin waits *versus*, 129–130
 in SPool molecular dynamics implementation,
 390–391
 synchronizing with I/O thread, 139–140
 timed wait in C++11, 140–142
 Image-filtering algorithms, 401
 Immutability, 127
 Implicit memory fence, 122
 Implicit tasks, 228, 229, 255–256, 266
 Implicit thread pool, 225–226
 Instruction level parallelism, 10, 158
 Integration of equations of motion, 378–380
 parallel algorithm, 379–380
 Intel 64 architecture, CAS instruction, 171
 Intel 8086, 2–3
 Intel C++ Composer XO, 398–399
 Intel compilers, 491
 data alignment, 398–399
 vectorization, 392, 394, 397–399
 Intel Parallel Suite, 308
 Intel Sandy Bridge, 391–392, 393–394
 Intel Thread Building Blocks. *See* Thread Building Blocks
 Intel x86 architecture, 160
 CAS instruction, 171
 Intel Xeon Phi, 10, 12, 14, 24, 292, 301, 343, 365–366,
 391–392
 task recycling and, 484–485
 Internal control variables (ICVs), 229–230
 actions, 230
 checking, 250–252
 initial values, 230
 scope of, 230
 Inter-process communication, 19
 public mutexes, 104
 Inverse Fourier transforms, 426–427, 428
 I/O
 concurrent programming benefits and, 27
 dispatching in OpenMP, 254
 futures for waiting on, 149
 job submission to task for, 478–479
 overlapping computation and, 27
 promises for waiting on, 150–152
 spin waits and, 163–164
 synchronizing with, 139–140
 Iteration space, 312

J

Jacobi method, 403, 404, 406
 Java, mutexes in objects, 127
 Job cancellation, 219, 220
 Job requests, 351
 Job submission, 229
 by client threads, 476–481
 complex recursive jobs, 479–480
 generic interface for, 477–478
 to I/O tasks, 478–479
 by main thread, 476, 477*f*
 nested parallel jobs, 359–360
 parallel, 409–410
 testing, 356–357
 Joining threads, 31–32
 Pthreads, 35–36
 Windows, 43–45

K

Kernels, 12

L

L1 caches, 8–9
 L2 caches, 8–9
 L3 caches, 8–9
 Lambda expression, 334–335, 495, 500–503
 syntax, 501
 Lambda function syntax, 331
 Laplace equation, 401
 finite-difference discretization, 402, 402*f*
 relaxation methods, 403
 Last In, First Out (LIFO), 18
 Latency, 3, 8
 Lattice swaps, 420–421, 421*f*
 Lazy initialization, 54, 56
 Leapfrog method, 378, 379*f*
 operations of, 379–380
 Libraries, 491. *See also specific libraries*
 portable, 493
 Library functions, 23
 error handling and, 31–32
 reentrant versions, 85
 thread safety of, 84–85
 Life cycles of threads, 25–28
 LIFO. *See* Last In, First Out
 Load-store architectures, 8, 102
 Local variables, 21
 Locality, 8
 lock(), 105, 116

Lock-free algorithms, 125, 171–174
 as alternative to critical sections, 171
 AReduction class, 172–174
 as optimistic approach, 171
 updating global data item, 171–172
 lock_guard object, 117, 118
 lock_guard<T>, 116–117
 Loop counters, 39
 Loop fusion, 386, 397
 Loop vectorization, 394
 Loops
 automatic parallelization of, 225–226
 vectorization of, 394

M

Macromolecules, 377–378
 Macrotasking, 385, 392
 NPool, 409–411, 417–418
 OpenMP, 387–389, 417–418
 barrier synchronizations, 389
 parallel region for, 408–409
 stationary temperature distributions example, 408–409
 work-sharing among threads, 389
 synchronization overhead, 392–393
 main() function, 2, 17
 local variable allocation, 18
 in SPMD thread pool, 209–210
 stack memory addresses, 18
 Main memory, 8
 advanced reads, 157–158
 delayed writes, 157–158
 Main thread, 20, 21
 partial results communicated to, 73, 73*f*
 malloc() function, 17
 map clause, 293
 Map operation, 262, 264
 master directive, 235–237
 Matrix allocations, 396–397
 Memory
 bandwidth, 3
 cache, 8
 issues, 9
 performance comparison and, 393
 data alignment, 398–399
 data placement in, 298
 delayed writes, 157–158
 domains, 17
 latency, 3, 8
 locality, 8, 393
 main, 8
 organization of, 18*f*, 21–22, 21*f*
 performance, 3

Memory (*Continued*)

- reading data from, 8–9
- writing data to, 9
- Memory affinity, 3–4, 470
- Memory alignment, 395–396
- Memory coherency, 159–160, 221–222
- Memory consistency issue, 9, 155
- Memory consistency model, 155–160
 - acquire-release, 175
 - C++11, 175–176
 - flush directive, 165–166
 - OpenMP, 165–166
 - Pthreads, 164–165
 - relaxed, 176–177
 - sequential, 122, 156–157, 175
 - acquire-release compared with, 177–178
 - global order in, 177, 177f
 - memory models, 175
 - problems with, 157–160
 - sequential consistency, 156–157
 - problems with, 157–160
 - synchronizing thread operations and, 175–176
 - weak-ordering, 160–164
 - weak ordering models, 160–164
 - busy wait, 162–164
 - memory fences, 161
 - mutexes, 161–162
- Memory fences, 155–156, 161
 - busy wait, 162–164
 - explicit, 165
 - implicit, 122, 161
 - mutexes and, 161–162
 - Windows, 188
- Memory operations, memory coherency and, 221–222
- Memory ordering constraints
 - acquire-release model, 175
 - consume semantics, 179
 - relaxed model, 176
 - sequential consistency, 175
 - synchronizing thread operations and, 175–176
 - tbb::atomic<T>, 186–187
- Memory ordering control, 161
- Memory pages, 297–298
- Memory system, 7–9, 153
- Memory wall, 3, 7, 153
 - in GPUs, 13
- memory_order_acq_rel, 179–180
- Memory-ordering option, 168
- Message Passing Interface (MPI), 5, 7
 - decoupling, 342
 - funneled multithreading support, 370
 - hybrid MPI-threads programming model, 369–373
 - message overhead in, 380
 - molecular dynamics problem and, 380
 - multithreading libraries and, 370
 - processes in, 369
 - SPMD style in, 370
- Microsoft Visual Studio 2013, 50
- Microtasking, 225–226, 385
 - OpenMP, 386–387, 408–409, 417
 - pipelined successive over-relaxation, 441–442, 447
 - synchronization overhead, 392–393
 - TBB, 411–412, 418–419
- MIMD. *See* Multiple Instruction, Multiple Data
- Molecular dynamics, 375–378
 - mechanical model, 375–376, 377–378
 - OpenMP implementations
 - barrier synchronizations, 389
 - macrotasking, 387–389
 - microtasking, 386–387
 - work-sharing among threads, 389
 - parallel implementations for, 384–391
 - parallel performance issues, 391–393
 - sequential application for, 380–384
 - code listing for, 382–384
 - common data set, 381
 - input and postprocessing, 381–382
- SPool implementation, 390–391
- TBB implementation, 391, 481–484, 481f
 - parallel tasks, 482
 - particle trajectory computation, 483–484
 - task recycling and, 484–485
- Monte Carlo computation of π , 70–74
 - C++11 atomic operations, 169–170
 - explicit tasks for, 286–287
 - parallel_invoke algorithm version, 329–330
 - parallelizing with OpenMP directives, 246–248
 - with parallel for, 248–249
 - using auxiliary function, 249–250
- thread safety, 83–84
 - stateless generators, 87
 - TBB, 94–95
 - Windows services, 92
- Moore, Gordon, 5
- Moore’s law, 5, 6
- MPI. *See* Message Passing Interface
- Multicore processors, 5–7
- Multiple Instruction, Multiple Data (MIMD), 10–11
- Multiprocessor systems. *See also* Symmetric multiprocessor systems
 - distributed memory, 4–5, 4f
 - shared memory, 3–4, 4f
- Multitasking, 1, 3, 19–20

- Multithreaded output, ordering with SafeCout, 112–113
 - Multithreaded processes, 20–24
 - hyperthreading, 24
 - launching threads, 21
 - memory organization, 21–22, 21*f*
 - threads as lightweight processes, 22–23
 - threads calling same function, 23–24
 - Multithreaded servers, 27
 - Multithreading, 1, 7
 - FFT and, 425
 - funneled multithreading support, 370
 - MPI and, 370
 - Mutex locking, 103
 - atomic operations as alternative to, 121, 126
 - C++11, 116–117, 118
 - concurrent containers and, 125
 - named critical sections *versus*, 245
 - OpenMP critical sections, 114, 115
 - restricted, 125
 - TBB, 119–120
 - untied tasks and, 286
 - Mutexes, 106*t*
 - busy waits and, 162–163
 - C++11, 116–118
 - scoped lock class templates, 116–118
 - encapsulation and, 127
 - exceptions and, 117
 - fair *versus* unfair, 104
 - inter-process communication and, 104
 - in Java objects, 127
 - kinds of, 103–106
 - locking, 103, 161, 162
 - memory coherency and, 159–160, 221–222
 - memory fences and, 161–162
 - number needed, 126
 - OpenMP, 114–116
 - interface, 115
 - ownership, 103
 - private *versus* public, 104
 - processes sharing, 104
 - Pthreads, 106–111
 - mutex-spin-lock programming interfaces, 107–108
 - public mutexes, 104
 - rwlock mutex, 216–217
 - scalar product of two vectors, 108–111
 - queuing, 119
 - read-write, 105
 - recursive, 104–105
 - scalability, 119
 - shared (read-write), 105–106, 119
 - spin, 119
 - standard *versus* spin, 104
 - TBB classes, 118–121
 - timed, 105
 - try_lock() functions, 105
 - unlocking, 162
 - Windows, 104, 114
 - Mutex-spin-lock programming interface, 107–108
 - Mutual exclusion, 101, 102
 - best practices, 126–127
 - compound thread-safe operations, 127
 - encapsulation, 127
 - mutexes, 126
 - restricted scope of atomic operations, 126
 - need for, 102–103
 - OpenMP directives for, 243, 244–245
 - as pessimistic approach, 171
 - task-centric regions and, 258, 286
 - TBB tools, 309
- ## N
- Named critical sections, 245
 - mutex locking *versus*, 245
 - Nested parallel jobs, NPool operation, 353–354
 - submitting, 359–360
 - Nested parallel regions, 227–228, 229, 250–251, 251*f*, 255–256
 - enabling in OpenMP, 232
 - setting max level for, 232
 - worker thread and, 256
 - Nested parallelism, 344
 - new, 17
 - Nonoverlap rule, 485
 - Non-Uniform Memory Access (NUMA), 3–4
 - NPool, 223, 286, 343, 385
 - API, 345–346
 - examples, 355–365
 - computing area under curve, 364–365
 - parent-child synchronization, 360
 - quicksort, 365
 - running unbalanced tasks, 356–357
 - task suspension, 360–361
 - taskgroup operation, 360
 - testing job submission, 355–365
 - traversing directed acyclic graph, 361–364
 - features, 343–345
 - job submission and management, 344, 344*f*
 - testing, 356–357
 - macrotasking, 409–411, 417–418
 - OpenMP compared with, 355
 - operation of, 350–355
 - assessment of environment, 355

NPool (*Continued*)

- avoiding deadlocks, 352
 - mapping tasks to threads, 353
 - nested parallel jobs, 353–354
 - passing data to tasks, 354
 - running recursive tasks, 353
 - parallel jobs, 346–350
 - memory allocation best practices, 349–350
 - Task class, 343, 347–348
 - TaskGroup class, 348–349
 - running parallel routines in parallel, 365–369
 - client code, 367–369
 - main parallel subroutine, 366–367
 - scaling, 412
 - STL containers in, 350
 - Task class, 343, 347–348, 356, 357–358
 - TaskGroup class, 343–344
 - TBB compared with, 355
- NUMA. *See* Non-Uniform Memory Access
- NVIDIA Corporation, 13

O

- OBLock, 198, 438
- Offloading code blocks to accelerators, 292–297
- OMP_CANCELLATION environment variable, 288
- OMP_PLACES environment variables, 298, 299
- OMP_PROC_BIND environment variable, 298
- One-dimensional FFT routines, 427–428, 455–456
- OpenACL, 13
- OpenCL, 13
- OpenMP, 13, 14, 25, 29, 74–78, 225–227, 307–308, 341, 385, 491
 - API, 225–226
 - Architecture Review Board, 227
 - atomic directive, 121, 122
 - atomic operations, 122–124, 167
 - barrier directive, 243
 - behavior of, 252–255
 - configuring, 229–233
 - automatic parallelization control, 233
 - controlling parallel regions, 232–233
 - controlling program execution, 230–231
 - thread limits, 232–233
 - critical directive, 243, 244–245
 - critical sections, 244–245
 - mutex locking *versus* named, 245
 - naming, 245
 - data-sharing attributes clauses, 240–242
 - directives, 226
 - adding to parallel constructs, 246–248
 - parallel, 227
 - target, 292–293
 - target data, 293–294
 - dynamic thread pools, 227–228, 232–233, 342
 - examples
 - area under curve, 77–78, 277–282, 278*f*, 280*f*
 - barrier and taskwait synchronization, 273–274
 - computation of π , 75–76, 245–250
 - data transfer among tasks, 254–255
 - database search, 78–80
 - I/O operation dispatch, 254
 - parallel operation on container elements, 262–267
 - parallel quicksort algorithm, 282–285
 - recursive algorithms, 242
 - tracking task scheduling, 272–273
 - traversing directed acyclic graph, 267–272
 - execution model, 226, 227–229, 228*f*, 256–257
 - handshake synchronization, 433–435
 - ICVs, 229–230, 231*t*
 - actions, 230
 - checking, 250–252
 - controlling program execution, 230–231
 - initial values, 230
 - scope of, 230
 - macrotasking with, 387–389
 - memory consistency, 165–166
 - microtasking with, 386–387
 - mutexes, 114–116
 - interface, 115
 - nested parallel regions, 227–228, 229
 - enabling, 232
 - setting max level for, 232
- NPool compared with, 355
- offloading code blocks to accelerators, 292–297
- OMP_CANCELLATION environment variable, 288
- OSynchP<T> template class, 203
- parallel and work-sharing constructs, 245–255
 - adding directives to, 246–248
 - parallel for, 248–249
- parallel region, 64–65, 222
- parallel section examples, 252–255
- programming interfaces, 226
- resources, 227
- running codes, 493
- scaling, 412
- sequential consistency in, 122
- SPool utility difference from, 341–342
- synchronization directives, 243–245
- synchronization utilities, 192
- task API, 255–262
 - barrier and taskwait synchronization, 273–274

- best practices, 286–287
- directed acyclic graph traversal, 267–272
- event synchronizations and explicit tasks, 257–258
- examples, 262–285
- parallel operation on container elements, 262–267
- task directive and clauses, 258–260
- task synchronization, 260–261, 260*f*
- task-centric execution model and, 255–257
- taskgroup directive, 261–262
- tracking task scheduling, 272–273
- task directive, 239–240
- task scheduler, 412
- task-centric execution model
 - implementation of, 257, 257*f*
 - motivations for, 255–257
- thread affinity, 297–301
 - close policy, 300–301
 - hyperthreading and, 301
 - master policy, 301
 - spread policy, 299–300
- thread management, 74, 233–242
 - master and single directives, 235–237
 - parallel directive, 227, 229
 - tracking thread activity, 252
- thread numbers, 229
- thread pools, 223, 227–228
- threadprivate directive, 91–92
- vath library interoperability with, 191
- vectorization, 301–304, 385, 394–395
 - declare simd directive, 303–304
 - simd directive, 302–303
 - standard interfaces for, 395
- worker teams, 64–65
- work-sharing directives, 233–242
 - sections and section, 237–238
- OpenMP constructs, 74
- Operating system (OS), 1
- Operating system processes, 17
- OS. *See* Operating system
- OSynchP<T> template class, 203
- Overlapping computation, I/O and, 27
- Over-relaxed Gauss-Seidel method, 403

P

- packaged_tasks, 150
- Parallel constructs, 233
 - adding directives, 246–248
 - cancellation of, 287–292
 - cancel directive, 287–288
 - checking for requests, 288–289
 - parallel regions, 219, 289–290
 - taskgroup regions, 290–292
 - examples, 245–255
 - styles of, 245–250
 - parallel directive, 227, 229
- Parallel execution, 1, 25, 192–193, 192*f*
- parallel for work-sharing directive, 248–249, 266, 301, 385
 - microtasking and, 407–408
 - vectorization and, 386
- Parallel job submission, 409–410
- Parallel operation on container elements, 262–267
- Parallel processing, 6
 - concurrency for, 20
 - inside cores, 10–12
- Parallel regions, 64–65, 204, 222
 - cancellation of, 219, 289–290
 - in macrotasking, 408–409
 - nested, 227–228, 229, 250–251, 251*f*, 255–256
 - enabling, 232
 - setting max level for, 232
 - worker thread and, 256
 - SPool utility, 64
 - task recycling, 484–488
 - task-centric execution and, 257–258
 - tasks triggered inside, 255
 - teams construct, 295
 - thread-centric execution model and, 257
- Parallel routines, running in parallel,
 - 365–369
 - client code, 367–369
 - main parallel subroutine, 366–367
- Parallel sections, examples, 252–255
 - barrier directive behavior, 252–254
- Parallel subroutines, 343
- parallel_do algorithm, 310,
 - 328–329
- parallel_for algorithm, 309, 385
 - example of usage, 317–320
- parallel_for template function, 312
- parallel_for_each algorithm, 310, 327–328
- parallel_invoke algorithm, 310, 329–330
- Parallelism
 - control, 375, 414, 423
 - data, 375, 392, 423
 - general program structure, 384–385, 384*f*
 - vectorization and, 385
 - instruction level, 10, 158
 - nested, 344
- parallel_reduce algorithm, 309, 320–327, 418
 - domain decomposition strategy in, 421
 - examples
 - parallel search for minimum value, 324–327
 - recursive area computation, 321–324
- parallel_scan algorithm, 309–310

- parallel_sort algorithm, 310
 - Parent-child synchronization, 360
 - Park-Miller minimal standard generator, 86–87
 - Partial results
 - accumulation, 73
 - communication, 73, 73*f*
 - Past the end pointers, 282
 - Peak performance, 2–3
 - Performance
 - function objects, 497
 - implementation comparison, 391–393
 - pipelines, 451–452
 - Periodic boundary conditions, 425
 - Persistent internal state, 96, 98
 - PipeBL<T> class, 438, 438*r*
 - pipeline algorithm, 310
 - pipeline class, 445–446
 - Pipeline concurrency pattern, 423–424, 424*f*
 - Pipelines, 194, 431–433
 - boolean locks and, 438
 - circular array of working buffers, 431–432
 - classes for, 438–440
 - synchronization member functions, 439–440
 - for concurrency, 423–424, 424*f*
 - concurrent queues and, 438
 - data dependencies and, 440
 - examples, 440–444
 - FFT with, 431–433
 - circular array of working buffers, 431–432
 - with handshake synchronization, 432–435
 - with producer-consumer synchronization, 435–436, 437–438
 - with handshake synchronization, 432–435
 - performance considerations, 451–452
 - with producer-consumer synchronization, 435–436, 437–438
 - TBB, 444–451, 444*f*
 - filter interface class, 445
 - pipeline class, 445–446
 - PSortBB.C code, 450–451
 - SOR code, 446–447
 - SORstage class, 447–450
 - threads in, 414
 - threads rank indices in, 438, 439
 - vath library classes for, 438–440
 - worker thread organization, 423
- Pipes, 19
- PipeThQ<T> class, 438, 438*r*
- Placement policies, 298–299, 300
- Pointer arithmetic
 - atomic<T> operations and, 169
 - in quicksort algorithm, 282
- Pointers, 17
 - atomic, 172–173
 - function, 496
 - global, 22, 54, 56
 - SPool objects, 66–67
 - past the end, 282
 - smart, 117
 - stack, 18
- Poisson equation, 401–402, 414
 - finite-difference discretization, 402, 402*f*
 - relaxation methods, 403
- Portable libraries, 493
- POSIX Threads (Pthreads), 14, 20, 25, 29, 41, 491
 - barrier function calls, 252
 - basic thread management interface, 34–35
 - broadcast, 135
 - condition variables, 131, 132
 - interface, 132
 - wait protocol, 105
 - waiting on, 133
 - data types, 33–34
 - detachment state, 35–36
 - error reporting, 32–33
 - idle waits in, 132–135
 - atomic mutex lock and wait in, 134
 - predicate check on return, 134
 - synchronizing with I/O thread, 139–140
 - wait protocol, 133–134
 - waiting on condition variables, 133
 - waking up waiting threads, 135
 - joining threads, 35–36
 - keys service, 96
 - memory consistency, 164–165
 - mutexes, 106–111
 - rwlock, 216–217
 - scalar product of two vectors, 108–111
 - mutex-spin-lock programming interfaces, 107–108
 - public, mutexes, 104
 - scaling, 412
 - signals, 135
 - SPool implementation, 66
 - thread attributes, 31
 - thread cancellation service, 67
 - thread data types, 30
 - thread function, 34
 - thread local storage, 96
 - interfaces, 90
 - thread management, 36–41
 - using, 32–41
- Power dissipation, 5, 6
- PowerPC architecture, CAS instruction, 171
- #pragma ivdep, 397

- #pragma omp simd, 395
- #pragma simd, 395, 398
- #pragma vector [clauses], 398
- Private mutexes, 104
- Private variables, 21, 241
- Processes, 1–2, 17–20
 - MPI and, 369
 - multitasking and, 19–20
 - multithreaded, 20–24
 - hyperthreading, 24
 - launching threads, 21
 - memory organization, 21–22, 21*f*
 - threads as lightweight processes, 22–23
 - threads calling same function, 23–24
 - mutexes shared by, 104
 - operating system, 17
 - stack and, 18–19
 - switching overhead, 19–20
 - threads as lightweight, 22–23
- Producer threads, 212–213, 214–215, 435
- Producer-consumer paradigm, 193, 200, 212
- Producer-consumer synchronization, examples, 437–438
- Programming models, 7
 - execution models and, 25–26
 - shared memory, 25
- promises, 150
- Promises, 130, 147–152
 - I/O operations and, 150–152
 - std::promise<T> class, 150–152
- Protein folding, 377–378
- pthread_cond_broadcast(), 135
- pthread_cond_signal(), 135
- pthread_cond_t, 131
- pthread_equal(), 36
- pthread_exit(), 36
- pthread_join(), 40–41
- pthread_mutex_t, 106
- Pthreads. *See* POSIX Threads
- pthread_attr_t, 33
- pthread_create(), 33, 34
 - signature, 35
- pthread_self(), 36
- pthread_spinlock_t, 107
- pthread_t, 33–34
- pthread_t pthread_self(), 36
- Public mutexes, 104

Q

- Queues. *See also* ThQueue<T> class
 - concurrent, 438
 - double-ended, 351
 - task

- global, 460
- TBB, 460–461
- thread-safe, 193, 351
 - finite capacity, 436
 - in producer-consumer synchronization, 435–436
- Queuing mutexes, 119
- Quicksort, 283*f*
 - parallel, 282–285
 - NPool, 365
 - recursive task function, 284–285
 - sequential, 282, 283

R

- Race conditions, 31–32, 39, 103
 - CAS operations and, 171–172
 - lock-free algorithms and, 171
 - mutual exclusion and, 171
- RAII. *See* Resource Allocation Is Initialization
- Rand(), 87
- Rand class, 69
- Random number generators, 86–89
 - local C++ objects, 88–89
 - stateless, 87–88
 - thread local storage
 - C++11, 91
 - OpenMP threadprivate directive, 91–92
 - TBB, 94
 - Windows services, 92–94
- Range classes, TBB, 314–315
- Rank indices, 438, 439
- read clause, 122
- Reader threads, 221
- Reader-Writer locks, 194, 215–218
 - Pthreads rlock mutex, 216–217
 - TBB, 217–218
 - Windows slim reader-writer locks, 217
- Read-write mutexes, 105
- Ready pools
 - scheduling strategies, 461
 - TBB, 460, 461
 - thread-owned, 460
- Real matrices, 426–427
- Real-valued functions, 457
- Recursive, divide and conquer, 222
- Recursive algorithms
 - area under curve
 - blocking style, 277–280, 278*f*
 - nonblocking style, 280–282, 280*f*
 - TBB parallel_reduce, 321–324
 - OpenMP versions, 277–282
 - parallel search for minimum value, 324–327
 - quicksort, 282–285

- Recursive mutexes, 104–105
- Recursive task functions
 - computing area under curve, 277–279, 280–281
 - directed acyclic graph traversal, 361–362
 - quicksort, 284–285
- Recursive tasks, NPool running, 353
- Recycling tasks, 469–470, 484–488
- Reduction operation, 73
 - in `parallel_reduce`, 322–324
- Reduction<T>, 112
- Reentrant generators, 87
- Reentrant library functions, 85
- Reference counts, 464, 469
- Registers
 - CPU, 11
 - vector, 11–12
- Relaxation methods, 403
 - Gauss-Seidel method, 403
 - Jacobi method, 403, 404, 406
 - over-relaxed Gauss-Seidel method, 403
- Relaxed memory model, 176–177
- Release operations, 161, 162
- Resource Allocation Is Initialization (RAII), 117
- Restricted locking, 125
- RingBuffer class, 182, 183–184, 436
- Runtime routines, 225
- Runtime system, implicit threadpools and, 225–226
- RWlock class, 218–222
 - examples
 - database search emulation, 219–220
 - reader threads, 221
 - shared container access, 220–221
 - writer threads, 220–221
 - interfaces, 218
 - memory operations, 221–222
 - TBB version, 221
- rwlock mutex, 216–217

S

- SafeCounter utility class, 304–305
- SafeCout, 112–113
- SBLock, 438
- Scalability
 - of mutexes, 119
 - parallel implementation comparison, 392
 - stationary temperature distribution implementation comparison, 412
- Scaling
 - Dennard, 5, 6, 6*r*
 - NPool, 412
 - OpenMP, 412
 - Pthreads, 412
- Scatter policy, 300*f*
- `schedule` directive, 239
- Scheduling
 - ready pool strategies, 461
 - task stealing strategy for, 256
 - tasks, 256
 - tracking, 272–273
 - worker threads and, 259–260, 266
 - TBB, 330–332, 461–462
 - tracking task scheduling, 272–273
- Scoped lock class templates, 116–118
- Scoped lock classes, TBB, 119–121
- Scoped locking method, 116–117
- Sdram, 8
- `section` directive, 237–238
- `sections` directive, 237–238
- Separating hyperplane, 414, 420–421
- `seq_cst` clause, 122
- Sequential code, data organization in, 426*f*, 427*f*
- Sequential compiler optimizations, conflicts with, 158–159
- Sequential consistency, 122, 156–157, 175
 - acquire-release compared with, 177–178
 - global order in, 177, 177*f*
 - memory models, 175
 - problems with, 157–160
 - examples, 159–160
 - memory accesses not atomic, 157–158
 - sequential compiler optimization conflicts, 158–159
- Sequential containers, 125
- Sequential regions, 204
- Servers, multithreaded, 27
- `SetCancellationPoint()`, 78–80
- `set_value`, 150
- Shared containers, accessing, 220–221
- Shared locks. *See* Reader-Writer locks
- Shared memory multiprocessor systems, 3–4
 - CPU limits, 3
 - molecular dynamics problem and, 380
- Shared memory multithreaded programming, 7
 - decoupling MPI from, 342
- Shared memory programming model, 25
- Shared (read-write) mutexes, 105–106, 119
- Shared variables, 21, 241
 - encapsulating, 111–112
 - global variables compared with, 241–242
 - serialized access to, 110
- Signals, 19
 - Pthreads, 135
- SIMD. *See* Single Instruction, Multiple Data
- `simd` directive, 301, 302–303, 385
 - clauses, 302–303
- SIMD vectorization, 398

- single directive, 235–237
- Single Instruction, Multiple Data (SIMD), 10–11, 301
 - vector register usage and performance of, 420
 - vectorization and, 393–394
- Single Program, Multiple Data (SPMD), 63–64, 366
 - MPI and, 370
- Single-core optimizations
 - memory alignment, 395–396
 - vectorization, 394–395
- Slim read-write lock, 136, 217
- Smart pointers, 117
- SMP. *See* Symmetric multiprocessor systems
- Sockets, 6
- Software organization, 491–493
- SOR. *See* Successive over-relaxation method
- SORstage class, 447–450
 - members functions, 448–450
 - private data members, 448
- SP. *See* Stack pointer
- SpBarrier class, 203, 385
- SpBlock, 198
- Spin barrier synchronization, 203–206
- Spin barriers, 203
- Spin mutexes, 119
- Spin waits, 104, 130, 195
 - barriers, 390–391
 - Boolean locks implementing, 198–199
 - idle waits *versus*, 129–130
 - I/O tasks and, 163–164
 - in SPool molecular dynamics implementation, 390–391
- SpinLock class, 180–182, 203
- Spin-locks
 - Pthreads programming interface, 107–108
 - scalar product of two vectors, 110
- Split constructor, 314, 322
- SPMD. *See* Single Program, Multiple Data
- SPMD thread pool, 208–210
 - main() driving worker activity, 209–210
 - worker thread launching, 208–209
 - worker thread termination, 210
- SPool utility, 63–69, 341–343
 - CancelTeam() utility, 67–68
 - client thread role, 64–66
 - creating and using objects, 66–67
 - creating objects, 66
 - examples, 69–74
 - adding long vectors, 69–70
 - area under curve, 77–78
 - database search, 78–80
 - Monte Carlo computation of π , 70–74
 - molecular dynamics implementation with, 390–391
 - OpenMP difference from, 341–342
 - parallel region, 64
 - pool operation, 68–69, 68*f*
 - public interface, 65
 - running thread team, 66
 - thread pool, 208
 - thread safety issue, 74
 - ThreadRange() utility, 67
- Stack, 17, 18–19
 - continuation stealing and, 467
 - function calls and, 18, 19, 19*f*
 - size of
 - C++11 threads, 53
 - multithreaded processes and, 31
 - SPool objects, 66
 - threads, 18–19, 21
- Stack buffer, 18
- Stack pointer (SP), 18
- Standalone applications, 1
- Standard Template Library (STL), 194
- Stateless generators, 87–88
- static qualifier, 86
- Static variables, 86, 89, 286
- Stationary temperature distributions, 401–402
 - finite-difference discretization, 402
 - heataux.c, 404
 - heat.c, 404–406
 - sequential version, 404–406
 - parallel versions, 407–412
 - NPool macrotasking, 409–411
 - OpenMP macrotasking, 408–409
 - OpenMP microtasking, 407–408
 - TBB microtasking, 411–412
 - performance comparison, 412–413
 - relaxation methods, 403
- std::async(), 148, 150
- std::atomic<T> class, 167–168
 - interfaces, 168
 - member functions, 168
 - Monte Carlo computation of π with, 169–170
 - properties, 169
- std::chrono::duration, 81
- std::chrono::milliseconds, 81
- std::chrono::seconds, 81
- std::complex<T>, 429
- std::condition_variable, 131, 137
- std::condition_variable_any, 138
- std::future<T> class, 147–148
 - basic member functions, 148
- std::launch::deferred, 148
- std::lock_guard<T>, 117, 118
- std::mutex, 116

Stdout, ordering multithreaded output to, 112–113
 std::promise<T> class, 150–152
 std::recursive_mutex, 116
 std::recursive_timed_mutex, 116
 std::ref(), 148
 std::shared_future, 150
 std::this_thread, global functions, 54
 std::thread object, 51
 member functions, 54
 std::thread object T, 52, 53*f*
 std::timed_mutex, 116
 std::unique_lock, 138
 std::unique_lock<T>, 118
 std::vector<T>, 124
 Stencil codes, 401
 data dependencies in, 414
 STL. *See* Standard Template Library
 STL containers, 124
 NPool use of, 350
 Successive over-relaxation method (SOR), 414–417
 diagonal swap version, 421
 parallel implementations, 417–419
 OpenMP and NPool macrotasking, 417–418
 OpenMP microtasking, 417
 TBB microtasking, 418–419
 performance issues, 419–420
 pipelined version, 440–444, 441*f*
 sequential implementation, 415–417
 with white-black ordering, 414–415
 Successor tasks, 464
 Sustained performance, 3
 SWRLOCK, 217
 Symmetric multiprocessor systems (SMP), 3–4, 4*f*, 14, 225
 cache coherency problem in, 154, 154*f*
 parallel thread pools and, 365–366
 SynchP<T> template class, 199–203
 barrier synchronization compared with, 201
 examples, 201–203
 OpenMP version, 203
 public interface, 199
 using objects from, 199–201, 200*f*
 Synchronization directives, 243–245
 Synchronization operations, 160
 Synchronization overhead, 392–393, 452
 Synchronization points, 193
 Synchronization primitives, 101, 130
 idle wait, 129–130
 in vath, 191
 Synchronization utilities, 180–185
 Synchronizing thread operations, 174–180
 comparing sequential consistency and acquire-release,
 177–178

consume instead of acquire semantics, 179
 global order in sequential consistency, 177
 memory models and memory ordering options, 175–176
 memory_order_acq_rel, 179–180
 relaxed memory model, 176–177
 System wait. *See* Idle waits

T

target construct, 292–293
 target data construct, 293–294
 Task class
 NPool, 343, 347–348, 356, 357–358
 TBB, 462–469
 member functions, 488–490
 task directive, 239–240, 256–257, 258–260, 266
 general form of, 258–259
 Task functions, 209
 barrier synchronizations in, 389
 in macrotasking, 387–388
 recursive
 computing area under curve, 277–279, 280–281
 directed acyclic graph traversal, 361–362
 quicksort, 284–285
 Task pool, 359
 Task queues
 global, 460
 TBB, 460–461
 Task scope ICVs, 230
 Task stealing, 316, 317, 326, 327, 461–462
 Task stealing scheduling strategy, 256
 Task suspension, 353
 NPool mechanism for, 360–361
 Task synchronization, 260–261, 260*f*
 graph traversal and, 268
 OpenMP, 244
 Task-centric execution model, 192, 256*f*, 341
 motivations for, 255–257
 mutual exclusion in, 258, 286
 OpenMP implementation of, 257, 257*f*
 parallel regions and, 257–258
 TBB, 308–309
 threads in, 286
 task_group class, 331–332, 476
 examples, 332–339
 area computation, fixed number of tasks, 332–334
 area computation, non-blocking style, 336–337
 area computation, tasks wait for children, 334–336
 canceling group of tasks, 337–339
 TaskGroup class, 343–344, 348–349
 constructing object, 349
 example of operation, 360
 public interfaces, 348

- taskgroup directive, 261–262, 267, 280, 281–282
 - cancellation of, 288
- Taskgroup region, canceling, 290–292
- Taskgroup synchronization, 260*f*, 261
- Tasks
 - barrier among tasks, 274–277
 - barrier synchronization and, 257, 260, 260*f*
 - best practices, 286–287
 - continuation, 315, 316–317
 - data transfer among, 254–255
 - dependencies among, 274
 - dynamic, 351
 - explicit, 229
 - event synchronizations and, 257–258
 - Monte Carlo computation of π with, 286–287
 - task directive creating, 258–259
 - implicit, 228, 229, 255–256, 266
 - mapping to threads
 - in NPool, 353
 - TBB, 464
 - NPool, 347–348
 - passing data to, 354
 - OpenMP API for, 255–262
 - barrier and taskwait synchronization, 273–274
 - directed acyclic graph traversal, 267–272
 - examples, 262–285
 - implementing barrier among tasks, 274–277
 - motivations for, 255–257
 - mutual exclusion in, 286
 - parallel operation on container elements, 262–267
 - task directive and clauses, 258–260
 - task synchronization, 260–261, 260*f*
 - taskgroup directive, 261–262
 - tracking task scheduling, 272–273
 - recursive functions
 - computing area under curve, 277–279, 280–281
 - quicksort, 284–285
 - recycling, 484–488
 - empty tasks in, 486
 - scheduling, 256
 - tracking, 272–273
 - worker threads and, 259–260, 266
 - successor, 464
 - taskwait synchronizations in, 335
- TBB
 - allocating, 465–466
 - blocking on children, 472–473
 - continuation passing, 473–474
 - defining in task_group environment, 330–331
 - hierarchical relations among, 464–465
 - mapping to threads, 464
 - recycling, 469–470
 - task spawning paradigms, 466–467
 - threads running, 469
 - threadprivate variables and, 286–287
 - triggered inside parallel regions, 255
 - unbalanced, 357–358
 - untied, 286
- TaskWait, 354
 - parent-child synchronization via, 360
- Taskwait synchronization, 260, 260*f*, 273–274, 335
- TBarrier class, 203, 412
- TBB. *See* Thread Building Blocks
- tbb::atomic<T>, 185–187
 - memory ordering options, 186–187
- tbb::concurrent_bounded_queue<T>, 436
- tbb::pipeline, 440
- tbb::task_scheduler_init object, 310
- tbbvars.sh script, 491
- teams construct, 294–297, 295*f*
- ThQueue<T> class, 210–215, 438
 - consumer threads, 212–213, 214, 215
 - design requirements, 211
 - producer threads, 212–213, 214–215
 - in producer-consumer synchronization, 436
 - public interface, 211, 212
 - thread safety and capacity, 440
 - using, 211–213
- Thread affinity, 297–301
 - close policy, 300–301
 - hyperthreading and, 301
 - master policy, 301
 - spread policy, 299–300
 - TBB and, 470
- Thread attributes, 30–31
- Thread Building Blocks (TBB), 14, 25, 29, 50, 307–308, 341, 385
 - abstraction levels of, 307–308
 - atomic data types, 167
 - atomic operations, 167
 - blocking *versus* continuation passing examples, 470–474
 - blocking on children, 472–473
 - continuation passing, 473–474
 - main thread code, 470–472
 - CAS operation, 171
 - child stealing, 317
 - concurrent container classes, 125–126
 - content of, 308–310
 - continuation passing, 473–474, 475
 - critical section interface, 120
 - empty tasks, 480–481
 - fork-join pattern in, 315–317
 - task spawning paradigms, 466–467
 - function objects in, 495, 500

- high-level algorithms, 309–310
 - built-in range classes, 314–315
 - fork-join parallel pattern, 315–317
 - integer range example, 313–314
 - operation of, 312–317
 - parallel_for example, 317–320
 - parallel_for operation, 319–320
- high-level scheduler programming interface, 330–332
 - defining tasks, 330–331
 - task_group class, 331–332
- initialization, 310–312
- installing and configuring, 491
- job submission by client threads, 476–481
 - complex recursive jobs, 479–480
 - generic interface for, 477–478
 - to I/O task, 478–479
 - by main thread, 476, 477f
- memory affinity issues, 470
- microtasking with, 411–412, 418–419
- molecular dynamics implementation, 391, 481–484, 481f
 - parallel tasks, 482
 - particle trajectory computation, 483–484
 - task recycling, 484–485
- mutexes, 118–121
 - classes, 118–119
- mutual exclusion tools, 309
- NPool compared with, 355
- pipelining threads in, 444–451, 444f
 - filter interface class, 445
 - pipeline class, 445–446
 - PSortBB.C code, 450–451
 - SOR code, 446–447
 - SORstage class, 447–450
- Reader-Writer locks, 217–218
- ready pools, 460, 461
 - global task queue, 460
 - scheduling strategies, 461
 - thread-owned, 460
- running codes, 493
- RWlock class, 221
- scheduler
 - allocating tasks, 465–466
 - API, 462–469
 - blocking *versus* continuation passing examples, 470–474
 - job submission by client threads, 476
 - mapping tasks to threads, 464
 - recursive computation of area under curve, 474–475
 - Task class, 462–469
 - task construction, 462–463
 - task hierarchical relations, 464–465
 - using, 470–475
 - scoped locking method, 116–117
 - scoped_lock internal classes, 119–121
 - task class member functions, 488–490
 - task management classes, 462
 - task queues, 460–461
 - task recycling, 469–470, 484–488
 - protocol for, 486–488
 - task scheduler, 412, 459
 - algorithm for, 469
 - task spawning
 - best practices, 468–469
 - with continuation passing, 467–468
 - paradigms, 466–467
 - waiting for children and, 468
 - task stealing in, 461–462
 - task_group class, examples, 332–339
 - thread local classes, 94–96
 - thread pools, 223
 - scheduling algorithm, 461–462
 - structure of, 459–462, 460f
 - threads running tasks, 469
 - thread-safe queue container, 436
- Thread cancellation service, 67, 220
- Thread centric programming paradigm, 192
- Thread functions, 30
 - Pthreads, 34
 - Windows, 42–43
- Thread local storage (TLS), 86, 89, 90f, 98–99
 - C++11 thread_local keyword, 90–91
 - OpenMP threadprivate directive, 91–92
 - Pthreads, 90, 96
 - TBB classes for, 94–96
 - Windows services for, 92–96
- Thread number, 229
- Thread objects, scope of, 54
- Thread pools, 64, 192, 208, 222–223, 341
 - dynamic, 227–228, 232–233, 342
 - explicit, 342, 342f
 - implicit, 225–226
 - OpenMP, 223, 227–228
 - parallel, 343, 365–366
 - SPMD, 208–210
 - main() driving worker activity, 209–210
 - worker thread launching, 208–209
 - worker thread termination, 210
 - TBB, 223
 - structure of, 459–462
- Thread safe functions, 24
- Thread safety
 - compound operations, 127
 - container, 124–126
 - counters, 326

- library functions and, 84–85
 - random number generators, 86–89
 - local C++ objects, 88–89
 - stateless, 87–88
 - SPool, 74
 - thread local storage services, C++11 `thread_local`
 - keyword, 89, 90–91
 - thread scheduler and, 84
 - Thread scheduler, thread safety and, 84
 - Thread synchronization, 101–102
 - mechanisms for, 101
 - Thread-centric models, 227, 341
 - parallel regions and, 257
 - `thread_local` keyword, 90–91
 - `threadprivate` directive, 91–92, 286–287
 - `ThreadRange()` utility, 67, 70
 - Threads, 1–2
 - barriers synchronizing, 257
 - basic libraries, 29–30
 - basic management of, 30–32
 - calling same function, 23–24, 23*f*
 - consumer, 212–213, 214, 215
 - data placement in memory and, 298
 - error handling and, 31–32
 - execution flows, 20–21
 - hybrid MPI-threads programming model, 369–373
 - intrinsic rank, 39–40
 - joining, 31–32
 - Pthreads, 35–36
 - Windows, 43–45
 - launching, 21
 - life cycles of, 25–28
 - as lightweight processes, 22–23
 - main, 20, 21
 - management of, 29
 - checking activity of, 252
 - OpenMP, 74, 252
 - Pthreads interface for, 34–35
 - Windows, 45
 - management overhead, 64
 - mapping tasks to
 - in NPool, 353
 - TBB, 464
 - tracking, 252
 - multitasking and, 19–20
 - mutual exclusion need among, 102–103
 - as objects, 52–53
 - pipelining, 414
 - in TBB, 444–451, 444*f*
 - placement policies, 298–299
 - producer, 212–213, 214–215
 - rank indices of, in pipelining, 438, 439
 - running tasks, in TBB, 469
 - stack role in, 18–19
 - stacks, 21
 - synchronizing operations, 174–180
 - memory models and memory ordering options, 175–176
 - task scheduling and, 256
 - task-centric environment and, 286
 - transferring ownership, 60
 - work-sharing among, 389
 - Thread-safe programming, 83–84
 - Monte Carlo computation of π , 83–84
 - Thread-safe queues, 193, 351
 - finite capacity, 436
 - in producer-consumer synchronization, 435–436
 - Time durations, 81
 - Time slices, 25, 299
 - Timed locks, 106
 - Timed mutexes, 105
 - Timed trylocks, 105
 - Timed waits, 81, 133, 195
 - C++11, 137, 140–142
 - Windows, 136
 - `time_duration`, 81
 - Timer objects, 139–140
 - Timers, 193, 194–195
 - implementation, 195
 - public interface, 195
 - using, 195
 - TLS. *See* Thread local storage
 - `try_lock()`, 105, 106, 116
 - Two-dimensional FFT routines, 428, 454, 456
 - Two-dimensional heat propagation, 425–428
 - FFT routines for, 427–428
 - method of solution, 426–427
 - physical problem, 425
 - sequential implementation, 428–430
- ## U
- Unconditional wait, 197
 - Unfair mutexes, 104
 - `unique_lock<T>`, 116–117
 - `unlock()`, 105, 116
 - Unordered containers, 125, 126
 - Untied tasks, 286
 - `until()`, 81
 - update clause, 122

V

vath library, 29, 64, 69

- barrier function calls, 252
- barrier synchronization classes, 203–204
 - public interface, 204
 - using, 204
- Boolean locks, 195–199
- building and testing, 492–493
- classes, 494
- code organization, 194
- DMonitor class, 381–382
- OpenMP interoperability, 191
- pipeline classes, 438–440
- portability, 191
- Reduction<T>, 112–113
- synchronization primitives in, 191
- synchronization services, 101–102
- synchronization utilities, 191, 193–194
 - code organization, 194
- Timers, 194–195

Vector processing, activating, 393–396

Vector registers, 11–12, 420

Vectorization, 10–12, 301–304, 393–396

- compiler feedback for, 395
- data parallelism and, 385
- declare simd directive, 303–304
- directives for
 - Intel compilers, 396
 - OpenMP, 394–395
- Intel compilers, 392, 394, 397–399
 - data alignment, 398–399
- loop fusion and, 386
- of loops, 394
- SIMD, 398
- SIMD and, 393–394
- simd directive, 302–303

Vectors

- correlated Gaussian, 377
- scalar product of, 108–111

Virtual CPUs, 25

Virtual memory systems, 297–298

Visual Studio, 491, 493

VisualC++, 491

W

wait(), 138

Wait protocol, Pthreads condition variables, 133–134

wait_for(), 141–142

WaitForIdle(), 66

Waiting on a condition, 26

wait_until(), 137, 150

Weak-ordering memory models, 160–164

- memory fences, 155–156, 161
 - busy wait, 162–164
 - explicit, 165
 - implicit, 122, 161
 - mutexes and, 161–162
 - spin waits, 163–164
 - Windows, 188

Windows API, 41

- condition variables, 135
- memory fences, 188
- slim reader-writer locks, 217
- slim read-write lock, 199
- SWRLOCK, 217

Windows atomic services, 187–190

Windows NT, 41

Windows thread API, 20

Windows threads, 29, 491

- atomic operations, 167
- CAS operation, 171
- condition variables, 131, 132, 135–136
 - interface for, 135
- creating, 41–43
- CRITICAL_SECTION interface, 114
- data types, 41–43
- joining, 43–45
- managing, 45
- mutexes, 114
 - processes sharing, 104
 - public, 104
- Reduction<T>, 112–113
- thread attributes, 31
- thread creation functions, 41–42, 43–45
- thread data types, 30
- thread function, 42–43
- thread local storage interfaces, 90
- timed waits, 136
- using, 41–49

Windows TLS services, 92–96

Windows Vista, 41, 135

- slim read-write lock and, 136, 217

Worker teams

- C++11 thread library, 61–63
- SPool, 64–65, 66

Worker threads, 55, 222

- communicating partial results, 73, 73*f*
- deadlocks and, 352, 353
- nested parallel regions adding, 256
- pipeline organization of, 423
- in SPMD thread pool
 - launching, 208–209
 - termination of, 210

- task scheduling and, 259–260, 266
- Working buffers, circular array of, 431–432
- Work-sharing, 225–226
 - automatic, 392
 - manual, 392
 - parallel, 248–249, 375
 - among threads, 389
- Work-sharing constructs
 - examples of, 245–255
 - parallel for directive, 248–249, 266, 301, 385
 - microtasking and, 407–408
 - vectorization and, 386
- Write buffer, 9, 157–158
- write clause, 122
- Writer threads, 220–221