

Listings

- Listing 1.1 Simple array example using Intel Threading Building Blocks. — **5**
- Listing 1.2 Simple array example using Fortran array syntax. — **7**

- Listing 2.1 Classic “Hello World” example using the POSIX thread API. — **14**
- Listing 2.2 Classic “Hello World” example with C++ threads. — **15**
- Listing 2.3 Classic “Hello World” example using the OpenMP API. — **18**
- Listing 2.4 Worksharing loop in the OpenMP API. — **20**
- Listing 2.5 The OpenMP master, masked, single, and sections constructs. — **21**
- Listing 2.6 Using OpenMP tasks to print “Hello World”. — **24**
- Listing 2.7 Classic “Hello World” example using Intel Threading Building Blocks. — **25**
- Listing 2.8 OpenMP taskloop Matrix Multiplication. — **26**
- Listing 2.9 Mutual exclusion features in the OpenMP API. — **29**
- Listing 2.10 Implicit and explicit barriers in the OpenMP API. — **32**
- Listing 2.11 Calculation of Fibonacci numbers to illustrate waiting for child tasks. — **34**
- Listing 2.12 Traversal of a linked list with task barrier. — **35**
- Listing 2.13 OpenMP code with task dependences. — **36**
- Listing 2.14 Matrix-matrix multiplication using task dependences. — **38**

- Listing 3.1 Broken point-to-point channel. — **74**
- Listing 3.2 Broken point to point channel, x86_64 assembly code. — **74**
- Listing 3.3 Correct point-to-point channel. — **75**
- Listing 3.4 Fixed point-to-point channel x86_64 assembly code. — **76**

- Listing 4.1 TBB example code before outlining for parallel execution. — **96**
- Listing 4.2 TBB example code after outlining for parallel execution. — **97**
- Listing 4.3 TBB code fragments to store a task in the task pool. — **98**
- Listing 4.4 Source code before outlining for parallel execution. — **104**
- Listing 4.5 Code of Listing 4.4 after outlining the code of the parallel region. — **104**
- Listing 4.6 Code of Listing 4.4 after outlining a nested thunk function. — **105**
- Listing 4.7 Element-wise array summation with a worksharing construct. — **106**
- Listing 4.8 Compiler-generated loop-scheduling code for Listing 4.7. — **107**
- Listing 4.9 Element-wise array summation using a task-loop construct. — **108**
- Listing 4.10 Code for element-wise array summation targeting a SIMD machine. — **110**
- Listing 4.11 Assembly code of the array code of Listing 4.10. — **111**
- Listing 4.12 Sorted assembly code of the unrolled code of Listing 4.11. — **112**
- Listing 4.13 Jammed SIMD code of the unrolled code of Listing 4.12. — **113**
- Listing 4.14 Implementing the master construct. — **114**
- Listing 4.15 Implementing the single construct. — **114**
- Listing 4.16 OpenMP static tasks and corresponding code generation pattern. — **116**
- Listing 4.17 Transforming a task construct into a lambda expression. — **117**
- Listing 4.18 Transforming a task construct into low-level runtime entry points. — **118**
- Listing 4.19 GCC high-level intermediate code for Listing 4.4. — **121**
- Listing 4.20 GCC low-level intermediate code for Listing 4.4. — **122**
- Listing 4.21 The clang compiler’s simplified AST for the code in Listing 4.4. — **124**

- Listing 4.22 LLVM byte code as generated by the `clang` compiler for Listing 4.4. — **125**
- Listing 5.1 Detecting NUMA structure of a system using `libnuma`. — **131**
- Listing 5.2 Pinning threads to cores using the POSIX⁺ thread API. — **133**
- Listing 5.3 Required preservation of thread-local state in OpenMP code. — **143**
- Listing 6.1 Implementation of a simple LIFO queue. — **147**
- Listing 6.2 Simple abstract lock base class. — **158**
- Listing 6.3 Implementation of a Test-and-Set lock. — **159**
- Listing 6.4 The Test and Test-and-Set lock's `lock()` method. — **161**
- Listing 6.5 Implementation of a Ticket lock. — **162**
- Listing 6.6 Implementation of the Mellor-Crummey & Scott lock. — **164**
- Listing 6.7 Random exponential backoff. — **175**
- Listing 6.8 Thread-safe implementation of `std::map`. — **178**
- Listing 6.9 Speculative lock template. — **182**
- Listing 6.10 Implementation of the `single` construct (`__kmpc_single`). — **187**
- Listing 6.11 Floating-point addition with compare-and-swap. — **189**
- Listing 6.12 Assembly code for Listing 6.11. — **189**
- Listing 7.1 Barrier timing code — **198**
- Listing 7.2 Code for a barrier with atomic counter. — **207**
- Listing 7.3 Atomic Up-Down barrier code. — **208**
- Listing 7.4 All-to-All barrier code. — **211**
- Listing 7.5 Reduction idiom causing false sharing and serialization. — **222**
- Listing 7.6 Using OpenMP reductions to avoid false sharing. — **223**
- Listing 8.1 The `CanonicalLoop` class. — **235**
- Listing 8.2 Initialize the `CanonicalLoop`. — **235**
- Listing 8.3 GCC code for a dynamic loop schedule. — **238**
- Listing 8.4 Code to detect nonmonotonicity in a parallel loop schedule. — **240**
- Listing 8.5 Code to implement the blocked static schedule. — **241**
- Listing 8.6 Code to implement the block-cyclic static schedule. — **242**
- Listing 8.7 Code to implement the guided dynamic schedule. — **246**
- Listing 8.8 Code to implement the `monotonic:dynamic` schedule. — **248**
- Listing 8.9 Consuming local iterations for `nonmonotonic:dynamic`. — **250**
- Listing 8.10 Stealing iterations in the `nonmonotonic:dynamic` schedule. — **251**
- Listing 9.1 Most basic task descriptor. — **260**
- Listing 9.2 Interface to the task pool class. — **261**
- Listing 9.3 Linked-list task-pool implementation with LIFO semantics. — **262**
- Listing 9.4 Creating and storing a task in the task pool. — **264**
- Listing 9.5 Allocating and initializing task descriptors. — **265**
- Listing 9.6 Finding and executing a task from the task pool. — **266**
- Listing 9.7 Function for allocating and deallocating task descriptors. — **267**
- Listing 9.8 Array implementation of a task pool with LIFO semantics. — **269**
- Listing 9.9 Storing tasks and handling overflows in the task pool. — **271**
- Listing 9.10 Extending the `Thread` class with a per-thread task pool. — **272**
- Listing 9.11 Storing and finding a task using multiple task pools. — **273**
- Listing 9.12 Random task stealing. — **274**

- Listing 9.13 Task pool implementation using a double-ended queue (deque). — 275
- Listing 9.14 Opening and closing taskgroup regions. — 278
- Listing 9.15 Task descriptor with metadata for the parent-child relationship. — 280
- Listing 9.16 Initializing and storing tasks with support for `taskwait`. — 283
- Listing 9.17 Executing tasks while maintaining the child count. — 284
- Listing 9.18 Walking the line of ancestor tasks and cleaning up parent tasks. — 285
- Listing 9.19 Delayed task creation with task dependences. — 287
- Listing 9.20 NUMA-aware task stealing. — 294
- Listing 9.21 OpenMP code showing the affinity clause of the task construct. — 301

