

# List of figures

- Figure 1.1 Layers of a parallel runtime system. — 2
- Figure 1.2 Paradigms for parallel programming models. — 4
- Figure 1.3 Parallel programming models by categorized by memory architecture. — 4
- Figure 1.4 XKCD 927: “Standards” — 7
- 
- Figure 2.1 Fork/join model to spawn and terminate parallel execution. — 16
- Figure 2.2 Distributing 92 iterations with static schedule with chunk size eight. — 20
- Figure 2.3 Conceptual execution scheme for tasks using a task pool. — 25
- Figure 2.4 Visualization of the task dependences in Listing 2.13. — 37
- Figure 2.5 Visualization of task dependences for the MxM code in Listing 2.14. — 39
- Figure 2.6 Amdahl’s law speedup curves. — 42
- Figure 2.7 Amdahl’s law efficiency curves. — 42
- Figure 2.8 Amdahl’s law serial fraction limits. — 44
- Figure 2.9 Amdahl’s law with slowdown. — 46
- 
- Figure 3.1 Von Neumann processor architecture. — 49
- Figure 3.2 In-order execution of instructions in a simple core. — 51
- Figure 3.3 Pipelined execution processing of instructions. — 52
- Figure 3.4 Out-of-order execution of instructions. — 55
- Figure 3.5 Branch prediction in an out-of-order execution engine. — 59
- Figure 3.6 Superscalar, parallel pipeline with multiple execution ports. — 60
- Figure 3.7 Two-way SMT pipeline (extended register file not shown). — 62
- Figure 3.8 Output of `hwloc-1s` for an Intel Core i7-6670HQ processor. — 63
- Figure 3.9 Examples of SIMD instructions of the Intel AVX-512 instruction set. — 65
- Figure 3.10 Four fully connected processor packages and their local memory. — 69
- Figure 3.11 Dual-socket Intel Xeon Platinum 8260L Processor system. — 70
- Figure 3.12 Example of a lost write. — 78
- Figure 3.13 Cache line state transitions. — 80
- Figure 3.14 The MESI protocol in action. — 81
- Figure 3.15 Effect of line placement on half round-trip time. — 84
- Figure 3.16 Effect of core placement on write time. — 85
- Figure 3.17 Effect of degree of sharing on store time. — 86
- Figure 3.18 Time for last poller to see a store. — 87
- 
- Figure 4.1 Stages of a compiler with front-end, middle-end, and back-end. — 91
- Figure 4.2 Example of a simplified abstract syntax tree. — 92
- Figure 4.3 Layers of a typical OpenMP runtime library. — 100
- Figure 4.4 Compiler with a middle-end for parallel code transformation. — 100
- Figure 4.5 Example of the simplified AST for an OpenMP parallel loop. — 101
- Figure 4.6 AST of Figure 4.5 with split parallel constructs and explicit clauses. — 102
- Figure 4.7 Recursive binary loop splitting to create tasks for parallel loop. — 109
- 
- Figure 5.1 Memory layout of a multi-threaded process. — 136
- Figure 5.2 Conceptual structure of a memory allocator (from [37]). — 136

- Figure 5.3 Storing the allocation directory in memory chunks. — 137
- Figure 6.1 LL-SC performance vs. performance of native atomic operations. — 149
- Figure 6.2 Uncontended and contended lock timelines. — 156
- Figure 6.3 The effect of polling interference. — 160
- Figure 6.4 Lock overhead (sum of lock and unlock time in thread). — 166
- Figure 6.5 Contended lock total machine throughput (single lock). — 167
- Figure 6.6 Contended lock total machine throughput (eight locks). — 167
- Figure 6.7 TTAS lock-reclaim rate. — 168
- Figure 6.8 Uncontended lock overhead (including `std::mutex`). — 169
- Figure 6.9 Contended lock total machine throughput (one lock). — 170
- Figure 6.10 Contended lock total machine throughput (eight locks). — 171
- Figure 6.11 Futex RILO (Root In, Last Out) time. — 174
- Figure 6.12 Contended lock machine throughput with backoff (one lock). — 176
- Figure 6.13 Contended lock machine throughput with backoff (eight locks). — 177
- Figure 6.14 Speculative lock total machine throughput. — 184
- Figure 6.15 Atomic `float32` addition machine throughput. — 190
- Figure 6.16 Mean required number of atomic operations. — 191
- Figure 7.1 Barrier imbalance. — 195
- Figure 7.2 LILO and LIMO times. — 197
- Figure 7.3 Atomic vs. store half round-trip time. — 202
- Figure 7.4 Naive broadcast Root In Last Out time. — 203
- Figure 7.5 Improving broadcast time by overlapping writes. — 204
- Figure 7.6 RILO time for different line broadcast widths. — 205
- Figure 7.7 Atomic Up-Down barrier performance. — 209
- Figure 7.8 All-to-All atomic barrier performance. — 210
- Figure 7.9 Communication in a hypercube barrier. — 212
- Figure 7.10 Communication in a Dissemination barrier. — 214
- Figure 7.11 Dissemination barrier performance. — 214
- Figure 7.12 Fixed and Dynamic tree mean communications on LIRO path. — 218
- Figure 7.13 Possible radix-4 fixed and dynamic binary trees for ten threads. — 218
- Figure 7.14 Fixed and dynamic tree LIRO times. — 219
- Figure 7.15 Fixed and dynamic tree LBW4 LILO times. — 219
- Figure 7.16 Barrier LILO time (including LLVM's OpenMP barrier). — 227
- Figure 8.1 Packing seven chunks of work onto three threads. — 230
- Figure 8.2 Maximum efficiency when executing  $n$  equal chunks on ten threads. — 230
- Figure 8.3 Different static allocations of seven equal chunks to three threads. — 232
- Figure 8.4 Atomics required for a guided schedule on 32 threads. — 245
- Figure 8.5 Contended atomic increment scaling on the Arm processor. — 247
- Figure 8.6 Parallel efficiency of loop scheduling (Square). — 254
- Figure 8.7 Parallel efficiency of loop scheduling (Increasing). — 254
- Figure 8.8 Parallel efficiency of loop scheduling (Random). — 255
- Figure 9.1 Example of task stealing respecting the hardware hierarchy. — 293
- Figure 9.2 Stack scheduling of two tasks with task A being suspended. — 297