On the realizability of hardware microthreading. Revisiting the general-purpose processor interface: consequences and challenges
Poss, R.C.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Listings

7.1 Two unrelated threads. ................................................. 127
7.2 Two unrelated threads with a race condition. ....................... 128
7.3 Independent ordering of loads/stores to different addresses. .... 128
7.4 Implementation of \texttt{twoprint}, insufficiently synchronized. .... 129
7.5 Proper synchronization for \texttt{twoprint}. ........................ 129
7.6 Invalid busy waiting for a value. .................................. 129
7.7 Invalid busy waiting on a pointer. ................................ 130

8.1 Code fragment using multiple virtual “global” channels. ............ 145
8.2 Translation of listing 8.1 to use only one “global” channel. ....... 145

9.1 Code sequence to access TLS on UTLEON3. .......................... 160

10.1 Automatic serialization code for listing H.21 (simplified). ......... 168
10.2 Generated assembly code for listing 10.1. ............................ 169

11.1 Concurrent SL code for the Livermore loop 3 (inner product). .... 179
11.2 Concurrent SL code for the Livermore loop 3, optimized. ......... 180

13.1 FORTRAN code for the Livermore loop 7. ........................... 205
13.2 Sequential C code for the Livermore loop 7. ........................ 206
13.3 SAC code for the Livermore loop 7. ................................ 206
13.4 Concurrent SL code for the Livermore loop 7. ....................... 207

E.1 Placement computation that extracts the size from a virtual cluster address. 266
E.2 Placement computation that extracts the absolute address of the first core in a virtual cluster. ........................................ 267
E.3 Placement computation that divides the current cluster in two and addresses either the upper or lower half. .......................... 267
E.4 Placement computation to place all the created thread at a core offset $P$ within the local cluster. ........................................ 267
E.5 Placement computation that divides the current cluster in two and addresses the other half relative to the current core. .................. 268
E.6 Placement computation to place all the created thread at the next or previous core within the local cluster. .......................... 268

F.1 Const designator to a mutable object. ................................ 276
G.1 Example code using the “create” construct and separate thread function to scale a vector. ................................................. 279
G.2 Example thread program using a channel interface specification. .... 280
G.3 Example use of the “innerprod” thread program. .......................... 282
G.4 Using the “address of” operator on a channel endpoint. ................. 284
G.5 Pointer aliasing a channel endpoint. .................................. 287
G.6 Implicit endpoint type conversion. ...................................... 288
G.7 Multiple orderings of the same endpoint names. ......................... 288
G.8 Example program fragment using “create” ............................ 290
G.9 Example program fragment using the “create” construct. .............. 292

H.1 Hand-crafted thread program. ........................................ 294
H.2 Hand-crafted C code. .................................................. 294
H.3 Alpha assembly generated using GNU CC. ................................ 295
H.4 C code with strategically placed macro uses. .......................... 296
H.5 Macro definitions for thread programs. ................................ 296
H.6 Externally instrumented Alpha assembly. ................................ 297
H.7 Automatically edited Alpha assembly. .................................. 298
H.8 Edited Alpha assembly with fewer used local registers. ............... 298
H.9 Macro definitions necessary for different interface arities. ............ 299
H.10 Code using multiple channel endpoints. ................................ 300
H.11 Generated assembly source for the “scal” thread program. ......... 300
H.12 Source code for the “innerprod” thread program. ..................... 303
H.13 Generated assembly for the “innerprod” thread program. ............ 303
H.14 Prototype “create” construct. ...................................... 304
H.15 Example use of “createsync.” ....................................... 304
H.16 Generated assembly from listing H.15. ................................ 305
H.17 Example use of “createsync” with omitted parameters. ............. 305
H.18 Generated assembly for listing H.17. .................................. 305
H.19 Support for “global” channel endpoints in “createsync.” ............ 306
H.20 Extension of listing H.19 for “shared” channels. .................... 306
H.21 Example vector-vector product kernel. ................................ 307
H.22 Generated code for listing H.21, using the “fused” creation interface. 308
H.23 Generated code for listing H.21, using the “detached” creation interface. 310
H.24 Syntax expansions for a sequential schedule. ........................ 311
H.25 Substitution for SPARC’s save. ..................................... 314
H.26 Substitution for SPARC’s restore. ................................... 314

J.1 QuickSort benchmark in SL, classic algorithm. .......................... 334
J.2 QuickSort benchmark in SL, families of two threads. .................. 335

K.1 Computation kernel executed by each logical thread. ................. 339
K.2 Workload implementation using an even distribution. ............... 339
K.3 Workload implementation using a round-robin distribution. .......... 340