Taming the complexities of the C11 and OpenCL memory models
—TECHNICAL APPENDIX—

John Wickerson[1] and Mark Batty[2]

[1] Imperial College London
[2] University of Cambridge

Abstract. We present in §A a change to the wording of the C11 specification that accommodates our proposed simplifications. We present the full C11 memory model (without the 'consume' memory order) in §B. We present and explain the full OpenCL memory model, including barriers, in §C.

A Proposed change to the C11 specification

Section 3.2 presented our proposal for simplifying the sequential consistency axioms in the C11 model. We give here our suggestion for how the specification document can be rephrased to accommodate our proposal, while maintaining the prose style used throughout the rest of the document.

Specifically: paragraphs 6–10 of [12, §7.17.3] can be removed and replaced with the following three:

1. A value computation A of an object M *reads before* a side effect B on M if A and B are different operations and B follows, in the modification order of M, the side effect that A observes.
2. A `memory_order_seq_cst` operation A is *SC-before* a `memory_order_seq_cst` operation B if there are operations X and Y such that the following all hold:
   (a) X reads before Y or X happens before Y or X precedes Y in modification order,
   (b) A either is equal to X or is a fence sequenced before X, and
   (c) B either is equal to Y or is a fence sequenced after Y.
3. There must be no cycles in *SC-before*.

B The C11 memory model in `.cat` – full version

We present in Fig. 1 the full C11 model (without the 'consume' memory order) in the `.cat` language.

Event labels:

$$\mathrm{W}_{\mathrm{na}}(l,v) \;\mid\; \mathrm{W}_o(l,v) \;\mid\; \mathrm{R}_{\mathrm{na}}(l,v) \;\mid\; \mathrm{R}_o(l,v) \;\mid\; \mathrm{RMW}_o(l,v_{\mathrm{old}},v_{\mathrm{new}}) \;\mid\; \mathrm{F}_o$$

where l ranges over memory locations, o over memory orders, and v over a set $\mathbb{V}$ of values assumed to contain at least 0 and 1.

Sets of events:
- R, W, F: read or RMW events, write or RMW events, fence events
- I: initialisation events (one non-atomic write to zero per location)
- naL: events accessing a non-atomic location
- na: events representing a non-atomic operation
- $o.rlx$, $o.acq$, $o.rel$, $o.ar$, $o.sc$: events parameterised by the respective memory order

Relations over events:
- thd: an equivalence relation over all (non-initialisation) events from the same thread
- loc: an equivalence relation over all (non-fence) events that access the same location
- sb (sequenced before): a partial order, contained in thd, depicting the order of instructions in the source code
- rf (reads from): contained in $W \times R$, relating writes to reads when the locations and values match, each read reads from exactly one write
- mo (modification order): a total order on writes to the same atomic location
- S: a total order on events in $o.sc$

(a) C11 events and relations

Synchronisation

1. $acq := (o.acq \cup o.ar \cup o.sc) \cap (R \cup F)$
2. $rel := (o.rel \cup o.ar \cup o.sc) \cap (W \cup F)$
3. $Fsb := [F] ; sb$
4. $sbF := sb ; [F]$
5. $rs' := thd \cup (unv ; [R \cap W])$
6. $rs := mo^? \cap rs' \cap \neg((mo \cap \neg rs') ; mo)$
7. $sw := ([rel] ; Fsb^? ; [W \setminus na] ; rs ; rf ; [R \setminus na] ; sbF^? ; [acq]) \cap \neg thd$

Happens-before and coherence

8. $hb := ((sb \cup (I \times \neg I)) \cup sw)^+$
9. $\mathbf{irr}(hb)$ (Hb)
10. $coh := (rf^{-1})^? ; mo ; rf^? ; hb$
11. $\mathbf{irr}(coh)$ (Coh)

Consistency of reads

12. $\mathbf{irr}(rf ; hb)$ (Rf)
13. $vis := (W \times R) \cap hb \cap loc \cap \neg((hb \cap loc) ; [W] ; hb)$
14. $\mathbf{irr}(rf ; [naL] ; \neg vis^{-1})$ (Narf)
15. $\mathbf{irr}(rf)$ (Rmw1)
16. $\mathbf{irr}(mo ; mo ; rf^{-1})$ (Rmw2)
17. $\mathbf{irr}(mo ; rf)$ (Rmw3)

Sequential consistency

18. $rb := (rf^{-1} ; mo) \setminus id$
19. $S_p := Fsb^? ; (rb \cup mo \cup hb) ; sbF^?$
20. $\mathbf{acyclic}(o.sc^2 \cap S_p)$ (Sc)

(b) The C11 consistency predicate. $consistent := (\mathsf{Hb}) \wedge \cdots \wedge (\mathsf{Sc8})$

21. $conflict := ((W \times W) \cup (W \times R) \cup (R \times W)) \cap loc$
22. $incl := thd \cup (\neg na)^2$
23. $dr := conflict \cap \neg hb \cap \neg hb^{-1} \cap \neg incl$
24. $\mathbf{empty}(dr)$ (Dr)
25. $ur := thd \cap conflict \cap \neg sb \cap \neg sb^{-1} \cap \neg id$
26. $\mathbf{empty}(ur)$ (Ur)

(c) The C11 faulty predicate. $faulty := (\mathsf{Dr}) \vee (\mathsf{Ur})$

Fig. 1. The C11 memory model (without 'consume') in `.cat`

C The OpenCL memory model in `.cat` – full version

We present in Fig. 2 the full OpenCL model (including execution barriers) in the `.cat` language. In the following, we discuss the changes that must be made to the model, which are highlighted in the figure.

Execution barriers and relaxed atomics When orchestrating communication between threads, via the global or the local memory, programmers must avoid *data races*. The traditional mechanism for avoiding data races in OpenCL is the *execution barrier*. No thread can proceed beyond a barrier if another in the work-group has not yet reached it. Barriers, in turn, present another opportunity for programmer error: *barrier divergence*. If we identify each dynamic (i.e. run-time) instance of a barrier by its position in the code together with the iteration count of each loop that contains it, then OpenCL defines *barrier divergence* as occurring when two threads reach different dynamic barrier instances.

New in OpenCL 2.0 are C11-style relaxed atomics. Unlike barriers, which only synchronise within a work-group, relaxed atomics allow threads in *different* work-groups to communicate, and enable fine-grained lock-free concurrent programming style. Following the OpenCL 2.0 specification, we define barriers in terms of fences: we represent a barrier as an *entry fence* followed by an *exit fence*, and then impose certain synchronisation axioms.

Event labels are extended to introduce entry and exit fences, which are used to represent execution barriers.

Barrier synchronisation occurs between the entry fence of one thread and the exit fence of another thread in the same work-group executing the same dynamic barrier instance.

Barrier divergence occurs when an entry fence has no exit fence to synchronise with (Bd).[3] The definition is slightly complicated by requiring the existence of another thread in the same work-group; that is, barrier divergence cannot occur if $Nt = 1$, even though there are no barrier synchronisation edges in this case.

[3] [13, 32/18–21]

Event labels (extending the grammar of Fig. 6a):

$$\mathrm{W}_{o,s}(l,v) \mid \mathrm{R}_{o,s}(l,v) \mid \mathrm{RMW}_{o,s}(l,v_{\mathrm{old}},v_{\mathrm{new}}) \mid \mathrm{F}_{o,s}(r) \mid \mathrm{entryF}_s(r,b) \mid \mathrm{exitF}_s(r,b)$$

where b identifies a dynamic barrier instance.

Sets of events (plus those from Fig. 6a):
- $entryF$, $exitF$: entry, exit fences

Relations over events (plus those from Fig. 6a):
- B: an equivalence relation on $entryF \cup exitF$ that partitions entry and exit fences according to the dynamic barrier instance from which they originate

(a) OpenCL events and relations

Synchronisation

1. $acq := (o.acq \cup o.ar \cup o.sc) \cap (R \cup F) \cup exitF$
2. $rel := (o.rel \cup o.ar \cup o.sc) \cap (W \cup F) \cup entryF$
3. $Fsb := [F] ; sb$
4. $sbF := sb ; [F]$
5. $rs' := thd \cup (unv ; [R \cap W])$
6. $rs := mo^? \cap rs' \cap \neg((mo \cap \neg rs') ; mo)$
7. $incl := thd \cap na^2 \cup wg \cap s.wg^2 \cup dv \cap s.dv^2 \cup s.all^2$
8. $sw(r) := ([r \cap rel] ; Fsb^? ; [W \setminus na] ; rs ; [r] ; rf ; [R \setminus na] ; sbF^? ; [acq \cap r]) \cap incl \cap \neg thd$
9. $bsw(r) := (entryF \times exitF) \cap B \cap \neg thd \cap wg \cap r^2$
10. $scf := o.sc^2 \cup (G \cap L \cap F)^2$
11. $gsw := sw(G) \cup bsw(G) \cup scf \cap sw(L)$
12. $lsw := sw(L) \cup bsw(L) \cup scf \cap sw(G)$

Happens-before and coherence

13. $ghb := (G^2 \cap (sb \cup (I \times \neg I)) \cup gsw)^+$
14. $lhb := (L^2 \cap (sb \cup (I \times \neg I)) \cup lsw)^+$
15. $\mathbf{irr}(ghb)$ (O-Hb-G)
16. $\mathbf{irr}(lhb)$ (O-Hb-L)
17. $coh(hb) := (rf^{-1})^? ; mo ; rf^? ; hb$
18. $\mathbf{irr}(coh(ghb))$ (O-Coh-G)
19. $\mathbf{irr}(coh(lhb))$ (O-Coh-L)

Consistency of reads

20. $\mathbf{irr}(rf ; (ghb \cup lhb))$ (O-Rf)
21. $vis(hb) := (W \times R) \cap hb \cap loc \cap \neg((hb \cap loc) ; [W] ; hb)$
22. $\mathbf{irr}(rf ; [G \cap naL] ; \neg vis(ghb)^{-1})$ (O-Narf-G)
23. $\mathbf{irr}(rf ; [L \cap naL] ; \neg vis(lhb)^{-1})$ (O-Narf-L)
24. $\mathbf{irr}(rf)$ (O-Rmw1)
25. $\mathbf{irr}(mo ; mo ; rf^{-1})$ (O-Rmw2)
26. $\mathbf{irr}(mo ; rf)$ (O-Rmw3)

Sequential consistency

27. $rb := (rf^{-1} ; mo) \setminus id$
28. $S_p := Fsb^? ; (rb \cup mo \cup hb) ; sbF^?$
29. $S_p := \neg(unv ; [o.sc \cap \neg(s.all \cup s.dv)] ; unv) \cap S_p$
30. $\mathbf{acyclic}(o.sc^2 \cap S_p)$ (O-Sc)

(b) The OpenCL consistency predicate. $consistent :=$ (O-Hb-G) $\wedge \cdots \wedge$ (O-Sc)

31. $conflict := ((W \times W) \cup (W \times R) \cup (R \times W)) \cap loc$
32. $dr := conflict \cap \neg(ghb \cup lhb) \cap \neg(ghb \cup lhb)^{-1} \cap \neg incl$
33. $\mathbf{empty}(dr)$ (O-Dr)
34. $bd := [entryF] \cap ((\neg thd \cap wg) ; unv) \setminus ((bar.sw(G) \cup bar.sw(L)) ; unv)$
35. $\mathbf{empty}(bd)$ (O-Bd)

(c) The OpenCL faulty predicate. $faulty :=$ (O-Dr) $\vee$ (O-Bd)

Fig. 2. The OpenCL memory model in `.cat`, with additions to include execution barriers highlighted.