A Formal Model for Portable, Heterogeneous Accelerator Programming
Errata

Zachary J. Sullivan and Samuel D. Pollard

Sandia National Laboratories, Livermore CA, USA
{spolla,zsulliv}@sandia.gov

1 Default Expression for Instantiating Memory Spaces

In order to prove soundness of the type system, we need to know that we initialize views with well-behaved values. To track this information, we added a default value expression to the declaration of views in the syntax.

$$S \in \mathit{Statement} ::= C; S \mid \texttt{ret} \mid \texttt{decl}\ x := E; S \mid \texttt{decl}\ x\ \texttt{in}\ \mu\ \texttt{with}\ E; S$$

The default value expression is evaluated when a view is declared and a base mapping for a view sets every index n to that value.

$$\frac{\pi \notin \mathrm{Dom}(\mathbb{M}(\mu)) \quad \langle\langle \mathbb{M} \mid\models L \parallel E \rangle\rangle \Downarrow V \quad M = \lambda n.\, V}{\langle\langle \mathbb{M} \parallel \mathbb{S} \mid\models L \parallel \texttt{decl}\ x\ \texttt{in}\ \mu\ \texttt{with}\ E; S \rangle\rangle \longrightarrow \langle\langle \mathbb{M}[\mu, \pi \mapsto M] \parallel \mathbb{S} \mid\models L[x \mapsto (\mu, \pi)] \parallel S \rangle\rangle}\ \mathit{GDeclView}$$

For typing this default value express, its type need only match the views base value type.

$$\frac{\Gamma, x{:}\texttt{view}(\mu, B_i) \vdash S \mathbin{@} \texttt{Host} \quad \Gamma \vdash E : B_i}{\Gamma \vdash \texttt{decl}\ x\ \texttt{in}\ \mu\ \texttt{with}\ ; S \mathbin{@} \texttt{Host}}\ \mathit{TDeclView}$$

2 Local evaluation of host in host-restricted environment

We need to run local host steps in the restricted machine memory environment that is accessible to it.

$$\frac{\begin{array}{c} C \in \{\texttt{decl}\ x := E, \texttt{set}\ x := E, \texttt{set}\ x(E_0) := E_1\} \\ \langle\langle \mathbb{M}|_{\mu \triangleright \texttt{Host}} \mid\models L \parallel C; S \rangle\rangle \longmapsto \langle\langle \mathbb{M}' \mid\models L' \parallel S \rangle\rangle \end{array}}{\langle\langle \mathbb{M} \parallel \mathbb{S} \mid\models L \parallel C; S \rangle\rangle \longrightarrow \langle\langle \mathbb{M}|_{\overline{\mu \triangleright \texttt{Host}}} \cup \mathbb{M}' \parallel \mathbb{S} \mid\models L' \parallel S \rangle\rangle}\ \mathit{GHStep}$$

3 Portability Theorem

First, portability is defined for an instantiation σ that cannot substitute host for free execution spaces. This is because the host is not seen as an accelerator, instead it is an orchestrater of programs executed on accelerators.

Definition 1 (Portable Program). *A program S is* portable *if and only if* $\text{Safe}(\text{Init}(S[\sigma]))$ *for a given set of execution and memory spaces, their accessibility relation, and any instantiation σ of its free execution and memory variables where $\chi[\sigma] \neq \texttt{Host}$.*

The portability theorem holds for a closed host program. Closed with respect to variables that would be stored in a type environment. There can be free execution and memory space variables in Δ.

Theorem 1 (Typing Ensures Portability). *If $\vdash_\Delta S \ @ \ \texttt{Host}$, then S is portable.*

4 Changes to Logical Predicates

In the process of mechanization, we also discovered issues with the logical relations given in the original appendix. We highlight the changes here, but to fully understand their implications, one must examine the full mechanization.

4.1 Globally and Locally Well-behaved Machine Memory Mappings

In the same manner as the operational semantics restricting to accessible machine memory mappings when evaluating within an execution space, the definition of well-formed machine memory mappings gets a global definition that holds for the host-level machine and a local one that holds when restricting that to specific execution space.

$$\begin{aligned}
\mathcal{M} = \{(\Psi, \mathbb{M}) \mid\ & \forall \mu, \pi.\, \pi \notin \text{Dom}(\mathbb{M}(\mu)) \implies \pi \notin \text{Dom}(\Psi(\mu)) \\
& \wedge \\
& \forall \mu, \pi, n.\, \pi \in \text{Dom}(\Psi(\mu)) \implies \\
& \quad \mathbb{M}(\mu)(\pi)(n) \in \mathcal{V}[\![\Psi(\mu)(\pi)]\!]\} \\
\mathcal{M}(\chi) = \{(\Psi, \mathbb{M}) \mid\ & \forall \mu, \pi.\, \pi \notin \text{Dom}(\mathbb{M}(\mu)) \wedge \mu \rhd \chi \implies \pi \notin \text{Dom}(\Psi(\mu)) \\
& \wedge \\
& \forall \mu, \pi, n.\, \mu \rhd \chi \wedge \pi \in \text{Dom}(\Psi(\mu)) \implies \\
& \quad \mathbb{M}(\mu)(\pi)(n) \in \mathcal{V}[\![\Psi(\mu)(\pi)]\!]\}
\end{aligned}$$

Note that another change here is that we additionally include that a free pointer in a machine memory mapping implies a free pointer in the machine memory typing. This is essential in proving that declaring a view preserves logical predicates (see the revised rule above). Therein, we learn that our program has taken a step with $\pi \notin \text{Dom}(\mathbb{M}(\mu))$ and we must propagate this information to our machine typing. This means that $\text{Dom}(\Psi) = \text{Dom}(\mathbb{M})$ for well-behaved machine memory mappings.

4.2 Step indexing of Statements Predicates

Despite not having higher-order state, our original logical predicates were not well-founded because they included an infinite inclusion chain. The common

approach of step-indexing broke this circularity. Note that this was not needed for the terminating sub-language of expressions. There is now an additional Kripke property that for any program with fewer steps, that these relations on statements holds.

$$\begin{aligned}\mathcal{X} = \{(n, \Psi, L, S, \chi) \mid \forall \Psi', \mathbb{M}.\, \Psi \sqsubseteq \Psi' \wedge (\Psi', \mathbb{M}) \in \mathcal{M}(\chi) \implies \\ \text{Final}(\langle\!\langle \mathbb{M} \mid\!\models L \parallel S \rangle\!\rangle) \\ \vee \\ \exists! XState.\, \langle\!\langle \mathbb{M} \mid\!\models L \parallel S \rangle\!\rangle \longmapsto XState \wedge \\ \exists \mathbb{M}', L', S'.\, XState = \langle\!\langle \mathbb{M}' \mid\!\models L' \parallel S' \rangle\!\rangle \wedge \\ \forall m < n.\, (\Psi', \mathbb{M}') \in \mathcal{M}(X) \wedge (m, \Psi', L', S', \chi) \in \mathcal{X}\}\end{aligned}$$

Another change in the (above) local and (below) global predicates is that we consider all accessible machine typings as well.

We needed to add step-indexing within well-behaved work queues because they contain well-behaved local statements. Note that this satisfies the Kripke property for fewer steps since it only contains the well-formed local statements for any n.

$$\mathcal{S} = \{(\Psi, \mathbb{S}) \mid \forall n, L, S, \chi.\, (L, S) \in \mathbb{S}(\chi) \implies (n, \Psi, L, S, \chi) \in \mathcal{X}\}$$

Finally, the global statements predicate is updated with a step, makes use of the new globally well-behaved machine memory space predicate, and we must ensure that any state that it steps to is well-behaved for some accessible machine memory typing. This latter point is in contrast to the local predicate, where we take an accessible machine memory typing as a predicate; globally, we actually have the ability of extending the typing.

$$\begin{aligned}\mathcal{G} = \{(n, \Psi, L, S) \mid \forall \mathbb{M}, \mathbb{S}.\, (\Psi, \mathbb{M}) \in \mathcal{M} \wedge (\Psi, \mathbb{S}) \in \mathcal{S} \implies \\ \text{Final}(\langle\!\langle \mathbb{M} \parallel \mathbb{S} \mid\!\models L \parallel S \rangle\!\rangle) \\ \vee \\ \left(\begin{array}{l} \forall GState.\, \langle\!\langle \mathbb{M} \parallel \mathbb{S} \mid\!\models L \parallel S \rangle\!\rangle \longrightarrow GState \implies \\ \quad \exists \Psi', \mathbb{M}', \mathbb{S}', L', S'.\, GState = \langle\!\langle \mathbb{M}' \parallel \mathbb{S}' \mid\!\models L' \parallel S' \rangle\!\rangle \wedge \\ \quad\quad \Psi \sqsupseteq \Psi' \wedge (\Psi', \mathbb{M}') \in \mathcal{M} \wedge (\Psi', \mathbb{S}') \in \mathcal{S} \wedge \forall m < n.\, (m, \Psi', L', S') \in \mathcal{G} \\ \wedge \\ \exists GState.\, \langle\!\langle \mathbb{M} \parallel \mathbb{S} \mid\!\models L \parallel S \rangle\!\rangle \longrightarrow GState \end{array} \right) \\ \}\end{aligned}$$