Fence Placement for Legacy Data-Race-Free Programs via Synchronization Read Detection

Andrew J. McPherson\textsuperscript{1} Vijay Nagarajan\textsuperscript{1} Sumit Sarkar\textsuperscript{2} Marcelo Cintra\textsuperscript{3}

University of Edinburgh\textsuperscript{1} University of St. Andrews\textsuperscript{2} Intel\textsuperscript{3}

\{ajmcperson, vijay.nagarajan\}@ed.ac.uk ss265@st-andrews.ac.uk marcelo.cintra@intel.com

Outline

\begin{itemize}
  \item Fence placement is required to ensure legacy parallel programs operate correctly on relaxed architectures.
  \item The challenge is to place as few fences as possible without compromising correctness.
  \item By identifying necessary conditions for a read to be an acquire we improve upon the state of the art for legacy DRF programs by up to 2.64x.
\end{itemize}

Our Approach

\begin{itemize}
  \item Legacy DRF: Programs that would be DRF but lack the required annotations.
  \item We do not seek to enforce Sequential Consistency for the general case; instead we seek to ensure data race freedom.
  \item In short, we guarantee SC behavior for accesses that are race free in the SC world.
\end{itemize}

Key Insight

\begin{itemize}
  \item We have identified two signatures and proven that at least one must be fulfilled for a read to be an acquire.
  \item This allows us to prune orderings identified by Delay-set analysis (or its approximations).
  \item Existing fence minimization techniques can then be applied.
\end{itemize}

Acquire Signatures

\textbf{Control Acquire:}
\begin{itemize}
  \item A read feeds its value to a predicate tested for in a branch in its forward slice.
\end{itemize}

\textbf{Address Acquire:}
\begin{itemize}
  \item A read provides the address value for a subsequent data access that the read (acquire) protects.
\end{itemize}

An Example

\begin{tabular}{|c|c|c|}
\hline
\textbf{Legacy DRF Code} & \textbf{Delay-set Fence Placement} & \textbf{Pruned Orderings Fence Placement} \\
\hline
\textbf{P1} & \textbf{P1} & \textbf{P1} \\
\hline
\textbf{a1 \ : \ x = b1 \ : \ sp1} & \textbf{(F1)} & \textbf{(F1)} \\
\hline
\textbf{a2 \ : \ y} & \textbf{(F2)} & \textbf{(F2)} \\
\hline
\textbf{a3 \ : \ flag = 1 \ : \ b1 \ : \ while(flag = 1)} & \textbf{(F3)} & \textbf{(F3)} \\
\hline
\end{tabular}

We assume alias analysis has determined that \textit{sp1} and \textit{sp2} may alias with both \textit{x} and \textit{y}, but not \textit{flag}.

Delay-set analysis detects the following critical cycles:
\begin{equation}
\begin{aligned}
(a1, a3, b1, a1), \quad (a2, a5, b5, a2), \quad (a1, a2, b2, b3, a1), \quad \text{and} \quad (a1, a2, b2, b3, a1).
\end{aligned}
\end{equation}

Our signatures allow \textit{a1} \rightarrow \textit{a2}, \textit{b1} \rightarrow \textit{b2} and \textit{b1} \rightarrow \textit{b2} to be pruned as none of \textit{a2}, \textit{b2} or \textit{b5} are acquires.

This means that \textit{F1}, \textit{F3}, and \textit{F5} are no longer required.

\textit{F2} and \textit{F4} are still required to prevent \textit{(a1, a2, b3, b1, a1)} and \textit{(a2, a3, b1, a2)}.

Sufficient Orderings for Correctness in DRF

\begin{itemize}
  \item \textit{r/w} \rightarrow \textit{w}_{rel}
  \item \textit{r/w} \rightarrow \textit{r/w}
  \item \textit{r}_{w} \rightarrow \textit{r/w}
  \item \textit{w}_{rel} \rightarrow \textit{r/w}
\end{itemize}

All reads and writes before the release (in program order) should be ordered before the release.

Prevalence of Acquire Types

\begin{itemize}
  \item We find that in practice acquires that meet the address signature also meet the control signature.
  \item To reinforce this point we performed an empirical study of 10 common synchronization primitives.
\end{itemize}

\begin{table}[h]
\begin{tabular}{|c|c|c|c|}
\hline
\textbf{Acquires} & \textbf{Addr} & \textbf{Cmd} & \textbf{Pure Addr} \\
\hline
Chase Lev WSLQ & $\checkmark$ & $\checkmark$ & $\times$ \\
Dijkstra & $\checkmark$ & $\times$ & $\times$ \\
Lin et al & $\checkmark$ & $\checkmark$ & $\times$ \\
McKee Locks & $\times$ & $\checkmark$ & $\checkmark$ \\
Peterson & $\checkmark$ & $\times$ & $\checkmark$ \\
Seymany & $\checkmark$ & $\checkmark$ & $\times$ \\
\hline
\end{tabular}
\end{table}

Implementation

We implement both variants of our approach:
\begin{itemize}
  \item \textbf{Fast}: Control acquires only
  \item \textbf{Safe}: Control and Address acquires as intraprocedural analyses in LLVM 3.4.1. We compare against Pensieve and a manual fence placement.
\end{itemize}

Escaping Reads Marked as Acquires

\begin{figure}[h]
\includegraphics[width=\textwidth]{performance_results.png}
\caption{Execution time with fences placed using Pensieve, Safe, Fast and manual fence placement.}
\end{figure}

Performance Results