Jotting down some of my thoughts about this paper, in case it's useful as supplemental material to the original work:

- How would TR work for accelerators? Ultimately, I see the biggest win of this approach on accelerators like GPUs, but that remains to be explored and evaluated. The wins could be significant because GPUs have large TLBs.
- Conceptually, it should be straightforward for TR to handle multiprogrammed workloads. We'd maintain separate lists of VMA sizes for them from an implementation point of view. But a detailed performance analysis remains to be done. (Thanks Sandhya Dwarkadas, for the discussion on this!)
- We've driven down TR's overheads to be pretty close to zero but I wonder whether we could do even better. Several years ago, when we worked on coalesced/clustered TLBs, Mark Hill had suggested that we consider OS migration tricks to more easily form contiguity/clusters for the small ranges of VPs that those schemes targeted. I still think that this could be a valuable approach over coalescing for an entire VMA if for some other workloads/system configurations, TR overheads rise. (Thanks Mark!)
- Direct segments, ranges, and DVM are go-to reading for me because they have given me the foundation to think about translation contiguity. I wonder whether there is scope to split a process's address space to pick a hybrid scheme between them and TR. (Segments and DVM are particularly aggressive in reducing TLB needs.) I also wonder whether the hybrid approach may look different depending on the composition of accelerators that the process invokes. Maybe this becomes particularly interesting with accelerator-level parallelism? [1]
- Are there things beyond TLBs for which translation contiguity (in general) and TR (more specifically) could be helpful?

[1] http://research.cs.wisc.edu/multifacet/papers/fastpath19_ALP.pdf