Transactifying Applications Using an Open Compiler Framework

Torvald Riegel

Joint work with

Pascal Felber, Christof Fetzer, Ulrich Müller, Martin Süßkraut, Heiko Sturzrehm

(Dresden University of Technology, Germany, and University of Neuchatel, Switzerland)

MICS

Problem

- How many real users does your STM have?
- Can real users easily try out your STM?

- Lack of users is a major problem
  - Lack of workloads
  - No education of users, no feedback from them

- Not just technical reasons (slow STM), also:
  - Availability, ease-of-use, lock-in?, costs, …

Our contributions

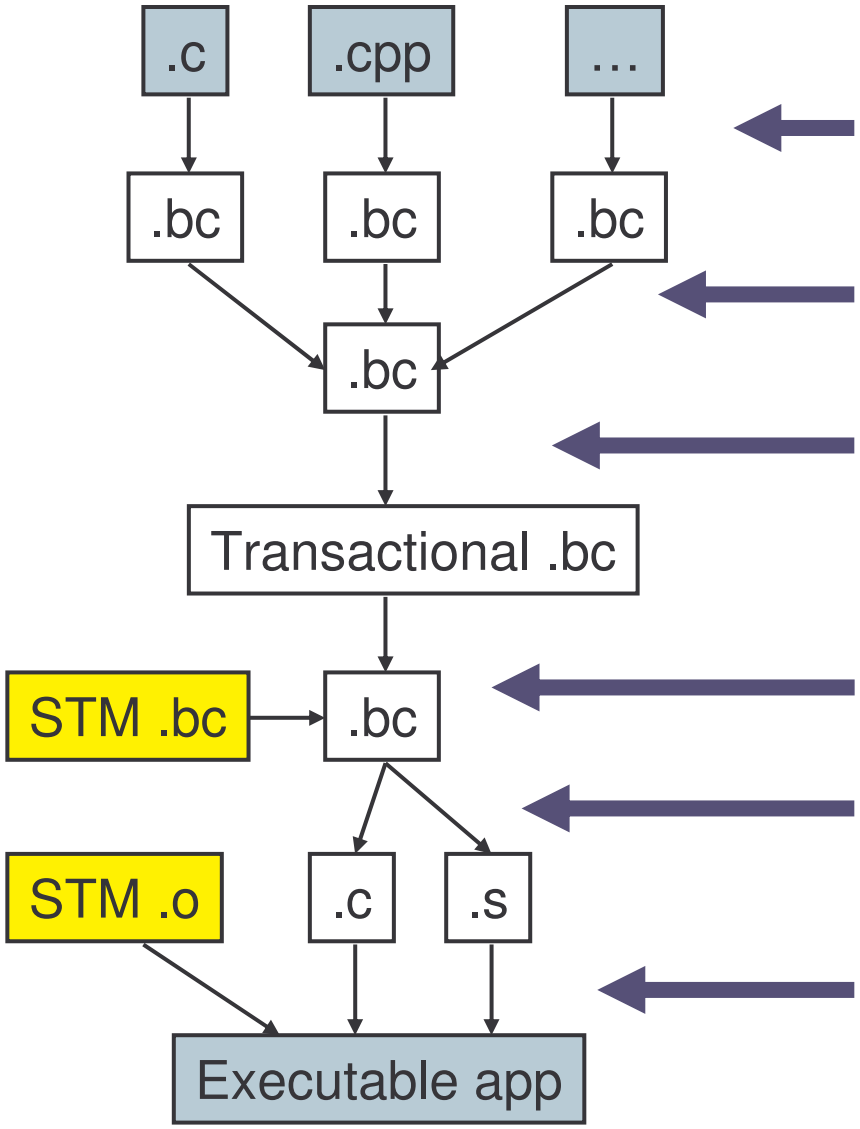
- **Tanger**
  - Open source compiler support
  - Application code with transaction boundary declarations is transformed to real transactional code that uses an STM
  - Uses LLVM's compiler framework
- **Tarifa**
  - Transforms declared transactions in IA32 assembler code to transactional asm code
  - More detail: ask me (later)
- License: GPL

What Tanger does

- Application code uses minimal declaration API:
  - begin, commit, init/shutdown function calls
  - Language syntax not changed, tools continue to work

- Tanger transforms code to use word-based STM API:
  - Find transactional code (bounded by begin/commit)
  - Redirect to STM functions (begin, commit, malloc, …)
  - Memory accesses to STM load/store
  - Redirect calls to transactional versions of functions

Usage (build cycle)



`llvm-gcc` (LLVM frontend):
- compile to intermediate representation (IR) (.bc = bitcode)

`llvm-ld` (LLVM linker/optimizer):
- link and optimize (potentially whole-program)

Tanger (compiler pass)
- **transform/create transactional code**

`llvm-ld`:
- link in STM, optimize

`llc` (LLVM backend):
- create target architecture asm or C code

`gcc`:
- compile and link with remaining binary objects or libraries

Why LLVM?

- LLVM (Low Level Virtual Machine):
  - Good intermediate representation (IR) for code (see next slide)
  - Very modular compiler
  - Link-time optimizations
  - JIT compilation support
  - Generates native code (e.g., x86, PowerPC, ARM, …) and C code (gcc)

- Alternatives:
  - Source to source translation: Easier? Optimizations?
  - gcc: Easier?

LLVM Intermediate Representation (IR)

- Sufficiently low-level (few dependencies), but still platform-independent and light-weight
- Can express C/C++ programs (important applications!)
- On-disk (.bc) and human-readable representations
- API for modifying IR is good and quick to learn
- IR uses types from source code (e.g., C structs)
- SSA
- Memory accesses are explicit (load/store)
- Stack contents in virtual registers unless accessed via pointers (fewer load/stores transformed!)

Current performance (tinySTM, single thread)

RBTree (STAMP)	Throughput	# calls in executable	
		Load	Store
Manual, gcc	11		
Manual, llvm-gcc	9.4	156	81
Tanger	8.8	127	77
Global lock, llvm	19		

Linked list	Throughput	# calls in executable	
		Load	Store
Manual, gcc	1.8		
Manual, llvm-gcc	1.3	16	3
Tanger	1.3	10	6

Things we'd like to build/have

- Our goals for Tanger: An STM environment:
  - Easy to use, practical: attract users!
  - Not just a research prototype
  - But allow plugging in research prototypes
- Compiler-based optimizations
- More features / subsystems:
  - Privatization, external actions, …
- STM support code (tracing, statistics, …)
- If you have ideas/plans, please talk to me

Benefits for users

- Decreases initial hurdles significantly
  - Minimal instrumentation overhead (parallelization, begin/commit, external actions)
  - Only have to select an STM
  - Performance is likely to get better, not worse

- Can get TM experience earlier
- Can follow research progress more easily

Benefits for research community

- User feedback: usability, workloads, …
- Smaller time-to-benchmark
- Environment for evaluation of your STM
- Comparing STMs gets easier (e.g., TL2 with tinySTM)
- Reference implementations for
  - Language integration proposals
  - Compiler support
- Benchmark writers could target Tanger instead of STM

Download Tanger, tinySTM, Tarifa at

http://tinystm.org