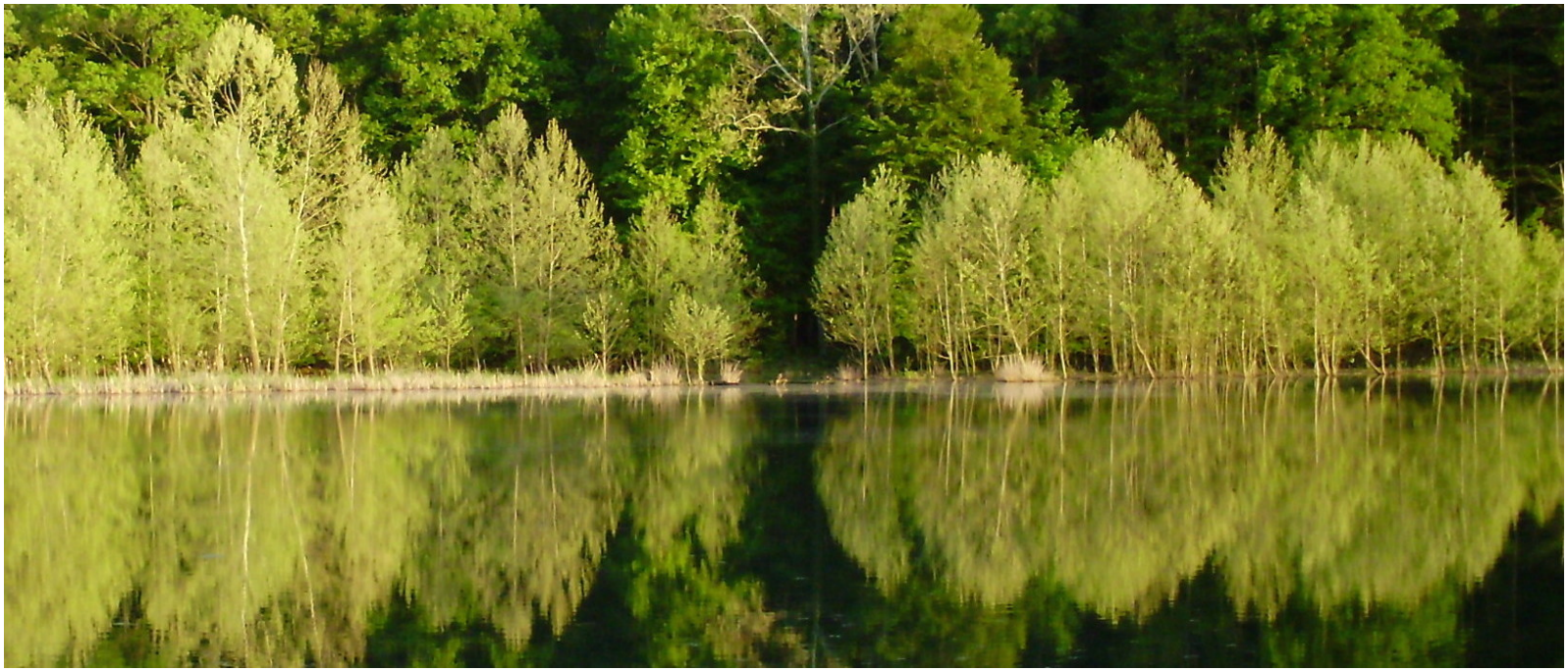
Jeremy G. Siek

Indiana University Bloomington

NII Shonan, May 2014



My Background

- 1998: Matrix Template Library (C++, generic programming, template metaprogramming, EDSL)
- 2002: Boost Graph Library (C++, generic programming)
- 2005: "Concepts" proposal for C++ (i.e. type classes).
- 2006: Gradual typing: mixing static and dynamic typing.
- 2009: Build-to-Order BLAS (BTO). (standalone DSL)
- 2010: Type-reflective metaprogramming, incremental type checking.
- 2014: The ParalleX execution model: asynchronous communication via active messages.
- 2014: Auto-tuning BTO via Monte Carlo Markov Chain methods.

My Position

- Staging is an important way to acheive high levels of *abstraction* and *performance*.
- Domain-specific languages (DSLs) are great for presenting reusable software at the appropriate level of abstraction.
- Standalone DSL's suffer the *language interoperability* problem, but *embedded* DSL's, to the most part, overcome this problem.
- Embedded DSL's traditionally have limitations wrt. abstraction leaks (e.g. error messages) and performance.
- Auto-tuning can deliver portable high-performance, but auto-tuners are difficult to build and historically have focused only on optimization parameters (e.g. unroll factors).

Questions

- Can we build reusable framworks for auto-tuning?
  - Reusable abstractions (containers and iterators).
  - Reusable optimizations (loop fusion, array contraction, tiling, data parallelism, task parallelism)
  - Can we interface generic static analyses and optimizations with domain-specific abstractions? (equational simplification, alias analysis, code motion, vectorization)
  - Efficient representations of the space of differently-optimized code variants.
  - Search algorithms.
- What can our general purpose languages do to support embedded DSLs?
  - E.g., fast stage zero performance
  - extensible type checking