

Tapir: Embedding Fork-Join Parallelism into LLVM's Intermediate Representation

January 2017 PPoPP '17

Pavas Handa M. F. R. Wani

Department of Computer Science
IIIT-D

Class Presentation, March 2017



INDRAPRASTHA INSTITUTE of
INFORMATION TECHNOLOGY DELHI

1 Foundations

- Introduction
- Background
- Motivation

2 Main Paper

- Contribution
- Related Work
- Implementation
- Result
- Conclusion

1 Foundations

- Introduction
- Background
- Motivation

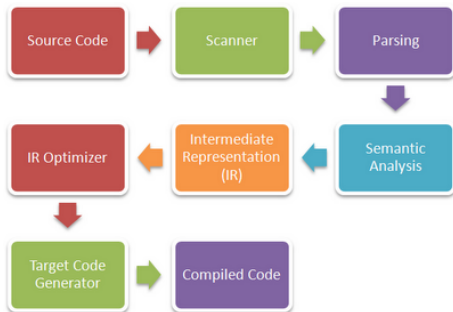
2 Main Paper

- Contribution
- Related Work
- Implementation
- Result
- Conclusion

LLVM: Low Level Virtual Machine

Is it yet another compiler ? whatever... Why should I care ?

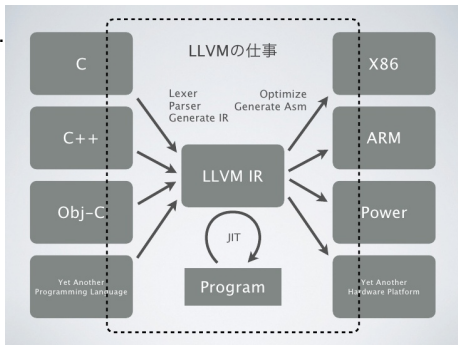
- Compiler Design 101:
 - Lexer (lex)
 - Parser (yacc)
 - Semantic Analyser
 - Intermediate Code
 - IR Optimizer
 - Target Code
 - Target Code Optimizer
- LLVM provides a **tool-chain**.
- Most important is **LLVM-IR**.
- LLVM is modular, unlike gcc.
- clang is a C compiler using LLVM.



LLVM-IR

How do I benefit ?

- Convert a language to LLVM-IR.
- Use either LLVM tools or your own.
- Sit back and let it work.
- You write the compiler only **once**.
- Any platform that supports LLVM, Your code Also runs (analogy JAVA)
- With a few function calls, enable state-of-the-art JIT support. (NO coding)
- Free optimizations! (No coding)



LLVM Pipeline¹

Image Scraped from Internet



INDRAPRASTHA INSTITUTE of
INFORMATION TECHNOLOGY DELHI

How does LLVM-IR look like ?

Simple MIPS like assembly with infinite registers

```
#include<stdio.h>

void
foo(unsigned e) {
    for (unsigned i = 0; i < e; ++i) {
        printf("Hello\n");
    }
}

int
main(int argc, char **argv) {
    foo(argc);
    return 0;
}
```

```
@str = private constant [6 x i8] c"Hello\00"

define void @foo(i32 %e) {
    %l = icmp eq i32 %e, 0
    br i1 %l, label %._crit_edge, label %.lr.ph

.lr.ph:                                ; preds = %.lr.ph, %0
    %i = phi i32 [ %2, %.lr.ph ], [ 0, %0 ]
    %str1 = getelementptr
                                   [6 x i8]* @str, i64 0, i64 0
    %puts = tail call i32 @puts(i8* %str1)
    %2 = add i32 %i, 1
    %cond = icmp eq i32 %2, %e
    br i1 %cond, label %.exit, label %.lr.ph

.exit:                                ; preds = %.lr.ph, %0
    ret void
}

define i32 @main(i32 %argc, i8** %argv) {
    tail call void @foo(i32 %argc)
    ret i32 0
}
```



1 Foundations

- Introduction
- **Background**
- Motivation

2 Main Paper

- Contribution
- Related Work
- Implementation
- Result
- Conclusion

Fork-Join Parallelism

Intrinsic support from compilers

- All mainstream compilers support fork-join parallelism.
- Provide support for Cilk Plus and OpenMP.
- Execution of Fork-Join creates a DAG.
- Execution usually via a Work Stealing Runtime.
- Provide Serial Semantics.

Serial Semantics

That is, the result of a parallel run is the same as if the program had executed serially. Serial semantics makes it easier to reason about the parallel application.



Optimization :(

Failure to optimize parallel code ![IR level]

- Compilers can't optimize parallel code efficiently.
- Due to implementation of parallel constructs.
- They don't understand parallel constructs !
- All major compilers **lower** parallel constructs to some primitives.
- No benefit from compiler optimizers. [IR level]

a

```
01 __attribute__((const)) double norm(const double *A, int n);
02
03 void normalize(double *restrict out,
04               const double *restrict in, int n) {
05     cilk_for (int i = 0; i < n; ++i)
06         out[i] = in[i] / norm(in, n);
07 }
```

b

```
08 __attribute__((const)) double norm(const double *A, int n);
09
10 void normalize(double *restrict out,
11               const double *restrict in, int n) {
12     #pragma omp parallel for
13     for (int i = 0; i < n; ++i)
14         out[i] = in[i] / norm(in, n);
15 }
```

a) Cilk Code b) OpenMP Code



INDRAPRASTHA INSTITUTE of
INFORMATION TECHNOLOGY DELHI

1 Foundations

- Introduction
- Background
- **Motivation**

2 Main Paper

- Contribution
- Related Work
- Implementation
- Result
- Conclusion

Parallel Encoding is Non Trivial

Making IR aware of parallelism is hard !

Use of serial code optimizers is dangerous for parallel programs.
It is not easy to encode Fork-Join parallelism in IR.

- Using Meta-data for annotation
- Intrinsic functions to determine parallel tasks
- A brand new IR with native parallel support
- Modification of existing IR



Challenges

Why can't we re-implement an new IR

- Introducing a new IR only work in academia, not Industry.
- Even if implemented, it would be painful, as it has complex dependencies.
- All previous approaches exhibited problems with serial optimizers.
- Elaborate Intrusive changes to existing compiler codebase is needed.



1 Foundations

- Introduction
- Background
- Motivation

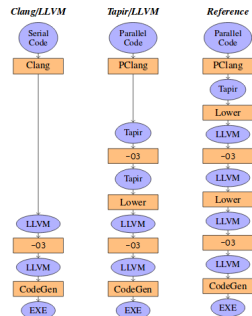
2 Main Paper

- Contribution
- Related Work
- Implementation
- Result
- Conclusion

TAPIR Contribution

Annotate existing IR with minimum changes

- Parallel aware IR represents fork-join parallelism asymmetrically, enabling serial & parallel optimizations.
- A mere 6000 lines addition to 4 Million lines of LLVM (0.15%)
- Implementation of parallel loop scheduling & elimination of unnecessary sync.
- Experiments and a reference implementation for benchmarks.



1 Foundations

- Introduction
- Background
- Motivation

2 Main Paper

- Contribution
- **Related Work**
- Implementation
- Result
- Conclusion

- Finding and Removing Unnecessary synchronization in Java.²
- Detection of instructions unaffected by parallel threads.³
- Modification of LLVM IR to support Open-SHMEM.⁴
- Inter-procedural analysis to perform various optimizations in critical Section Code ⁵

²ref [3]

³ref [26]

⁴ref [30]

⁵ref [6, 7]

- 1 Foundations
 - Introduction
 - Background
 - Motivation
- 2 Main Paper
 - Contribution
 - Related Work
 - **Implementation**
 - Result
 - Conclusion

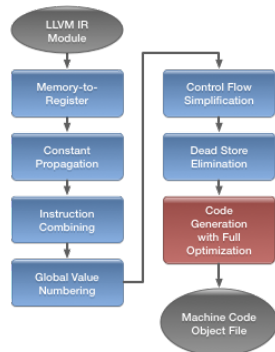
Implementation I

- Addition of 3 instructions to IR:
 - detach (async)
 - reattach (join)
 - sync (barrier)
- Minor differences with what we have seen.
- Advantages:
 - Easy introduction of Fork-Join in Compilers
 - Expressive IR to encode Fork-Join of different Languages.
 - Plays nicely with serial optimizers.
 - Performance gains in practical scenarios.



Implementation II

- The SSA form for LLVM IR must be adapted for Tapir.
- detach/reattach express parallel tasks asymmetrically both syntactically and semantically as in memory.
- Optimizations
 - Common sub-expression elimination
 - Loop-Invariant code Motion
 - TRE (recursion elimination)
 - Loop Scheduling.
 - many more



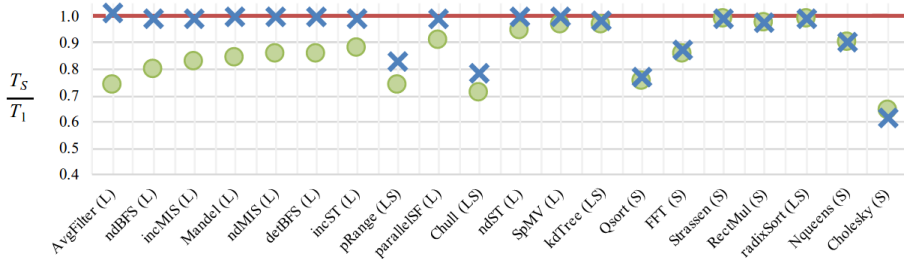
1 Foundations

- Introduction
- Background
- Motivation

2 Main Paper

- Contribution
- Related Work
- Implementation
- **Result**
- Conclusion

Result



O: reference compiler | X: tapir/llvm



INDRAPRASTHA INSTITUTE of
INFORMATION TECHNOLOGY DELHI

1 Foundations

- Introduction
- Background
- Motivation

2 Main Paper

- Contribution
- Related Work
- Implementation
- Result
- Conclusion

Conclusion

- TAPIR makes it very easy to write **Optimization Passes**.⁶
- Tapir enables FJP programs to benefit from both serial and parallel optimizations.
- Serial programs work on parallel constructs with little or no modification.

⁶<http://llvm.org/docs/Passes.html>



Questions

Interested people can opt for Advanced Compiler Optimization (Next Sem)



INDRAPRASTHA INSTITUTE of
INFORMATION TECHNOLOGY DELHI

For Further Reading I

-  A. V . Aho
Compilers: Principles, Techniques, and Tools.
Addison-Wesley, 2006.
-  LLVM Website.
www.llvm.org