Phoenix

Compiler & Tools Framework

Shahrokh Mortazavi
VC++
Microsoft

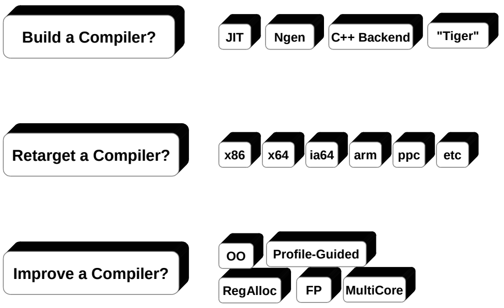
Agenda

- What is Phoenix?
- Demo : "ILdasm"
- Why Phoenix?
- Demo : IR-Viewer
- Phoenix Inside
- Demo : Tiger Compiler
- Status; Phoenix-Academic
- Links

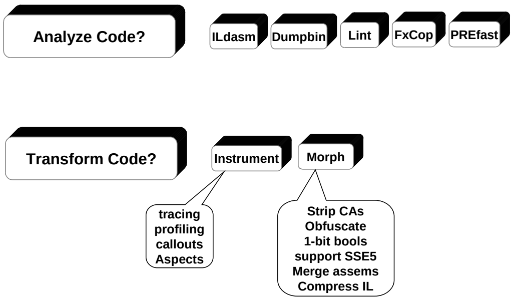
What is Phoenix?

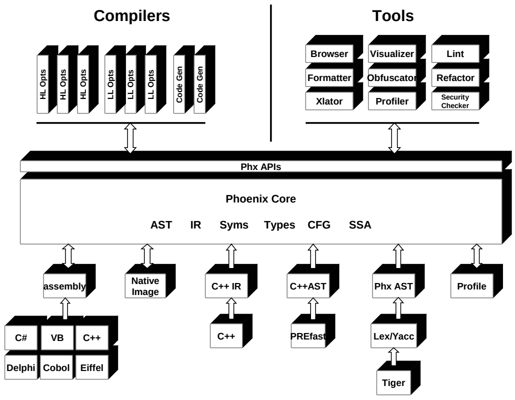
- Next-Generation Framework for
  - building Compilers
  - building Software Analysis Tools
- Mainline Microsoft compilers for 10+ years
- Framework for Academic Research & Teaching

For Compilers



For Software Tools





Demo

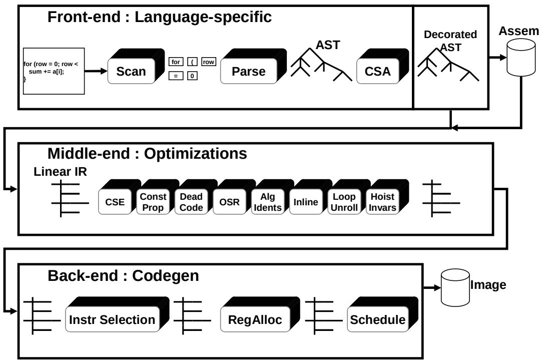
"ILdasm" using Phoenix

Let Phoenix do the heavy lifting

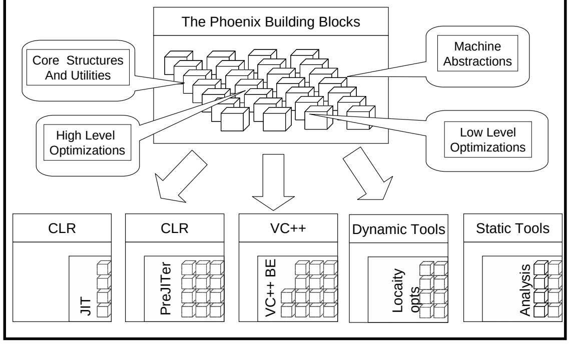
- ILdasm
  - < 1 kloc
- C2 (optimizing backend for C++)
  - 4 kloc
- Phoenix code base
  - total : 500 kloc

alias 15k	eh 27k	graphs 23k	ir 38k
metadata 19k	syms 14k	types 27k	x86 34k

  - samples :   20 kloc
  - doc :   9 MB   .chm



Pick & Mix



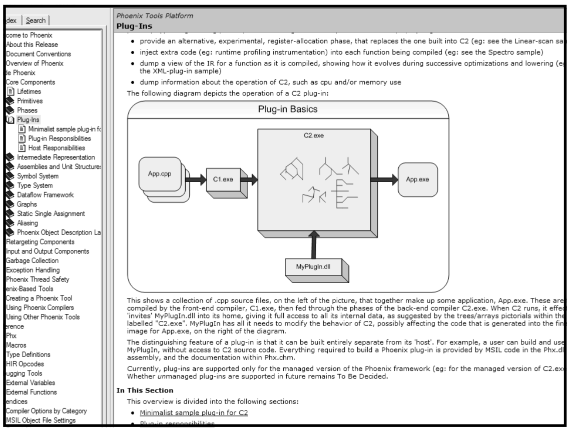
Why Phoenix (Microsoft)?

- Avoid duplicated effort
- Faster re-targetting
- Research⇔product xfer
- Academic

Why use Phoenix (Third Parties)?

- Robust, stable, documented, supported, long-lived, extensible
- Wide API
- JIT, Ngen ("PreJIT"), 'batch'
- Dual Mode -   .NET / native
- Rapid Retargetting
- Extensible IR (intermediate representation)
- Plug-ins; mix&match components
- Testbed for algorithm comparison
- Real-world Apps

This presentation is for informational purposes only. Microsoft makes no warranties, express or implied, in this summary.



Phoenix Tools Platform

Plug-Ins

- provide an alternative, experimental, register-allocation phase, that replaces the one built into C2 (eg: see the Linear-scan sa
- inject extra code (eg: runtime profiling instrumentation) into each function being compiled (eg: see the Spectro sample)
- dump a view of the IR for a function as it is compiled, showing how it evolves during successive optimizations and lowering (e the XML-plug-in sample)
- dump information about the operation of C2, such as cpu and/or memory use

The following diagram depicts the operation of a C2 plug-in:

This shows a collection of .cpp source files, on the left of the picture, that together make up some application, App.exe. These are compiled by the front-end compiler, C1.exe, then fed through the phases of the back-end compiler C2.exe. When C2 runs, it effe 'invites' MyPlugIn.dll into its home, giving it full access to all its internal data, as suggested by the trees/arrays pictorials within the labelled "C2.exe". MyPlugIn has all it needs to modify the behavior of C2, possibly affecting the code that is generated into the fin image for App.exe, on the right of the diagram.

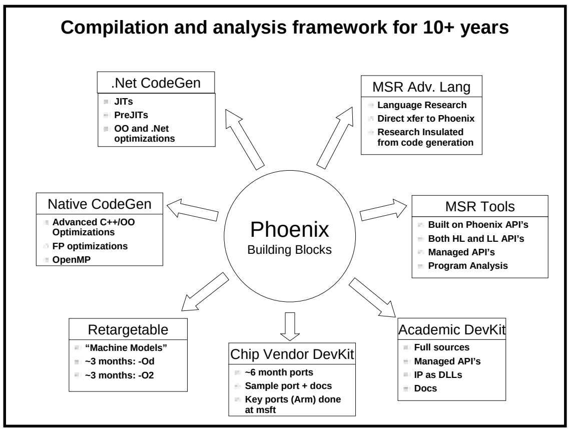
The distinguishing feature of a plug-in is that it can be built entirely separate from its 'host'. For example, a user can build and use MyPlugIn, without access to C2 source code. Everything required to build a Phoenix plug-in is provided by MSIL code in the Phx.dl assembly, and the documentation within Phx.chm.

Currently, plug-ins are supported only for the managed version of the Phoenix framework (eg: for the managed version of C2.ex Whether *un*managed plug-ins are supported in future remains To Be Decided.

In This Section

This overview is divided into the following sections:

- Minimalist sample plug-in for C2
- Plug-in responsibilities



Compilation and analysis framework for 10+ years

.Net CodeGen
- JITs
- PreJITs
- OO and .Net optimizations

MSR Adv. Lang
- Language Research
- Direct xfer to Phoenix
- Research Insulated from code generation

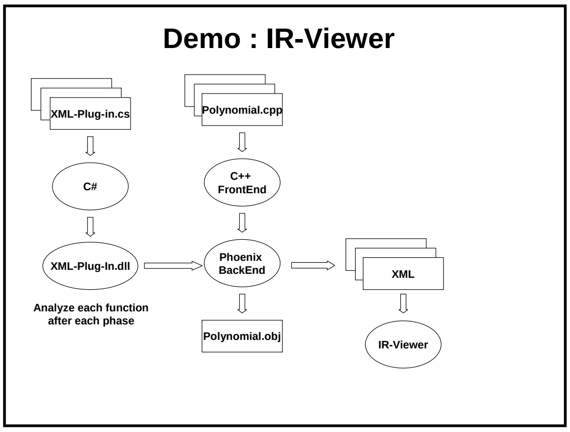
Native CodeGen
- Advanced C++/OO Optimizations
- FP optimizations
- OpenMP

MSR Tools
- Built on Phoenix API's
- Both HL and LL API's
- Managed API's
- Program Analysis

Retargetable
- "Machine Models"
- ~3 months: -Od
- ~3 months: -O2

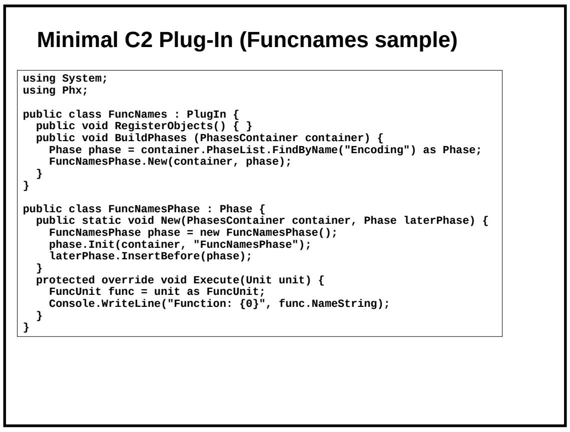
Chip Vendor DevKit
- ~6 month ports
- Sample port + docs
- Key ports (Arm) done at msft

Academic DevKit
- Full sources
- Managed API's
- IP as DLLs
- Docs



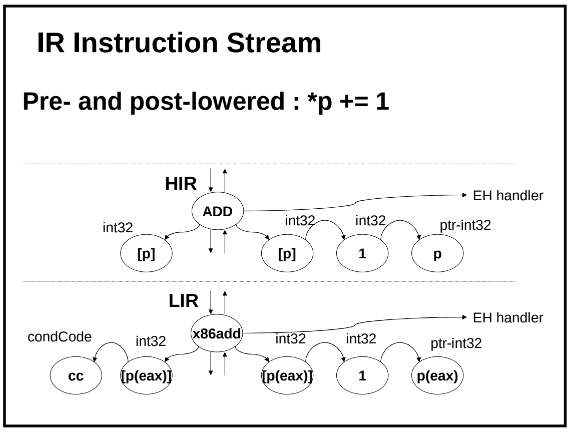
Demo : IR-Viewer

Analyze each function after each phase



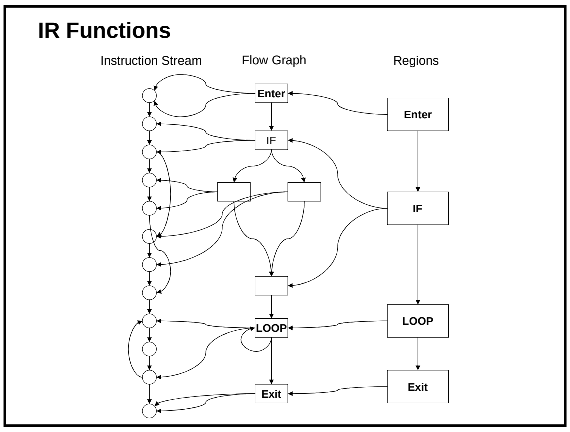
Minimal C2 Plug-In (Funcnames sample)

```
using System;
using Phx;

public class FuncNames : PlugIn {
  public void RegisterObjects() { }
  public void BuildPhases (PhasesContainer container) {
    Phase phase = container.PhaseList.FindByName("Encoding") as Phase;
    FuncNamesPhase.New(container, phase);
  }
}

public class FuncNamesPhase : Phase {
  public static void New(PhasesContainer container, Phase laterPhase) {
    FuncNamesPhase phase = new FuncNamesPhase();
    phase.Init(container, "FuncNamesPhase");
    laterPhase.InsertBefore(phase);
  }
  protected override void Execute(Unit unit) {
    FuncUnit func = unit as FuncUnit;
    Console.WriteLine("Function: {0}", func.NameString);
  }
}
```



IR Instruction Stream

Pre- and post-lowered : *p += 1



IR Functions

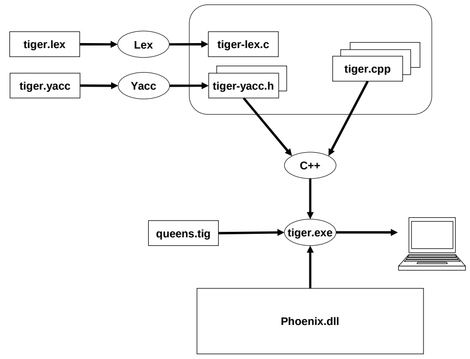
Demo : Tiger Compiler

- Tiger = small imperative language
- *Modern Compiler Implementation*, Appel [†]
- Builds AST and emits into Phoenix

[†] Object-Tiger, Fun-Tiger, PureFun-Tiger, Lazy-Tiger, Poly-Tiger

Example Tiger Program (8 queens)

```
let
  var  N := 8
  type intArray = array of int
  var  row := intArray [N] of 0; var col := intArray [N] of 0;
  var  diag1 := intArray [N+N-1] of 0; var diag2 := intArray [N+N-1] of 0
  function printboard() = (
    for i := 0 to N-1 do (
      for j := 0 to N-1 do print(if col[i]=j then " O" else " .");
      print("\n") ) )
  function try(c : int) =
    if c = N then printboard()
    else for r := 0 to N-1 do
      if row[r]=0 & diag1[r+c]=0 & diag2[r+7-c]=0 then (
        row[r]:=1; diag1[r+c]:=1; diag2[r+7-c]:=1;
        col[c]:=r; try(c+1);
        row[r]:=0; diag1[r+c]:=0; diag2[r+7-c]:=0
      )
in
  try(0)
end
```



Phoenix Status

- Builds Longhorn (NT)
- JITs .Net version of PowerPoint
- Basic optimizations coming online now
- Several tools plug-ed in
  - Code analysis; obfuscation, AOP, etc
- Academic program initiated

Phoenix Academic Program

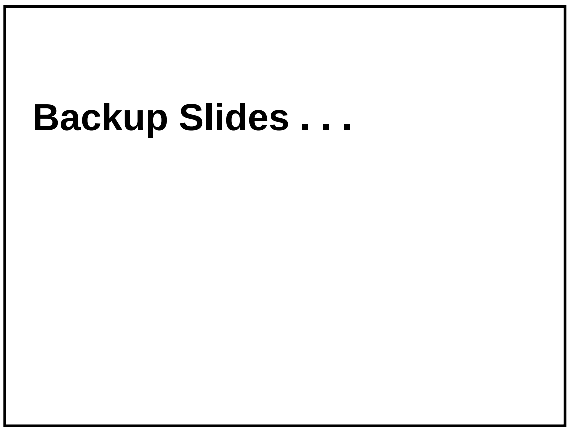
- Research and/or teaching
- Early access to Phoenix - ~100 copies so far
- Q&A Website; Updates

- Robust, stable, documented, supported, long-lived
- Dual-mode (unmanaged / managed)

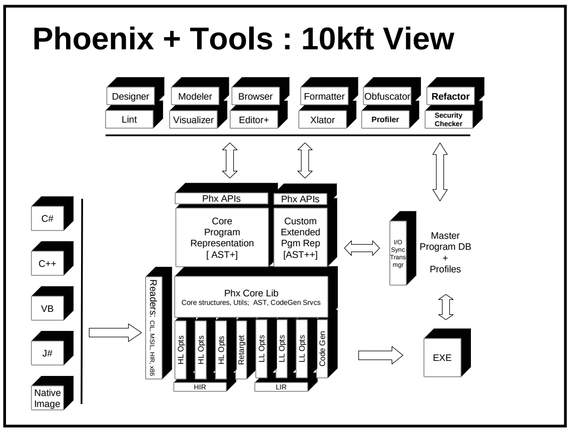
- RFPs : 40+ submissions; 12 awards
- DevLab in June 05

Links

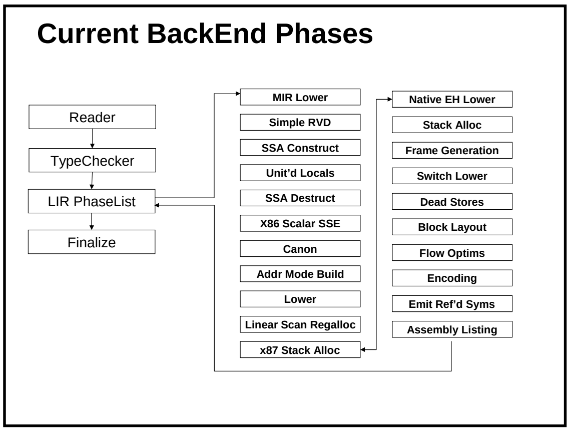
- http://research.microsoft.com/phoenix



Backup Slides . . .



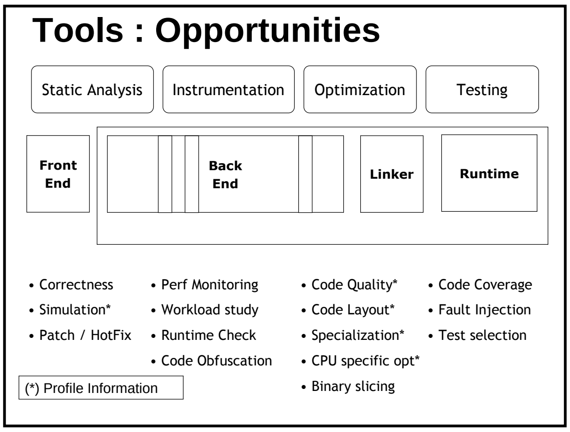
Phoenix + Tools : 10kft View



Current BackEnd Phases

Microsoft®
Your potential. Our passion.™

© 2004 Microsoft Corporation. All rights reserved.
This presentation is for informational purposes only. Microsoft makes no warranties, express or implied, in this summary.



Tools : Opportunities

- Correctness
- Simulation*
- Patch / HotFix

- Perf Monitoring
- Workload study
- Runtime Check
- Code Obfuscation

- Code Quality*
- Code Layout*
- Specialization*
- CPU specific opt*
- Binary slicing

- Code Coverage
- Fault Injection
- Test selection

(*) Profile Information