

# Open IMPACT

Wen-mei Hwu

Sanders-AMD Endowed Chair

University of Illinois at Urbana-Champaign

<http://www.gelato.uiuc.edu>

# Agenda

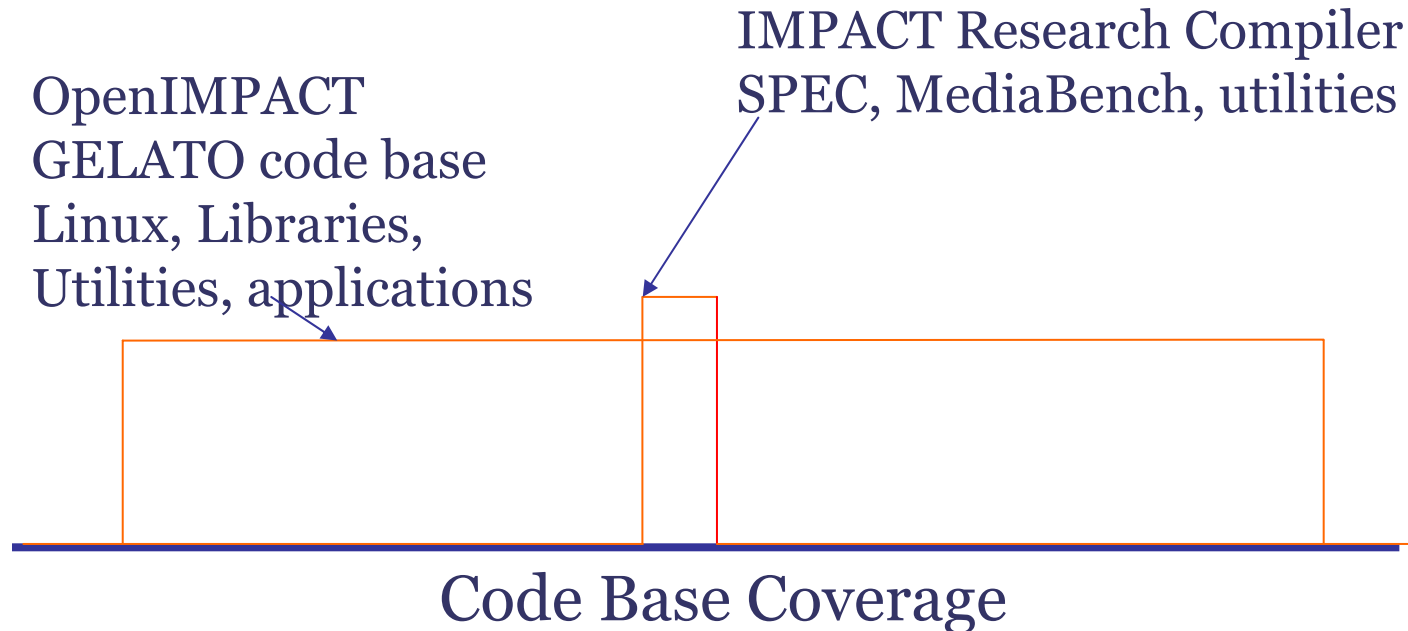
---

- Rationale, Goals, Status, Plans
- To solicit your active participation

# OpenIMPACT

- In 2002, the U. of Illinois released the IMPACT Compiler into the open-source domain as part GELATO ([www.gelato.org](http://www.gelato.org))
- OpenIMPACT encompasses many of the advanced compilation techniques developed by the IMPACT research team, including
  - programmatic logic analysis
  - predicated compilation
  - scalable interprocedural pointer analysis
  - instruction-level parallelism optimizations
  - profile-based optimization
  - speculative hyperblock acyclic and modulo scheduling
- Due to its heritage as a research compiler, OpenIMPACT is designed to achieve maximal output code performance with little concern for user interface, compile time and compiler memory usage.

# Main Challenge – widen the code base



We expect to define new research problems!

# The Current Model of Operation



The OpenIMPACT engineering team at UIUC contributes to the GELATO community by focusing on open source applications that are critical to GELATO members:

- The OpenIMPACT team has produced open-source applications and utilities executable binaries with record-setting performance on Itanium 2 platforms.
- OpenIMPACT also forms the main code base for future IMPACT research group
- OpenIMPACT team will work closely with GELATO members to make OpenIMPACT beta releases to Gelato Members starting 9/1/2004

# OpenIMPACT Team

- **John Sias** is the compiler architect for OpenIMPACT.
  - He is developing new ILP research compilation algorithms and future roadmap of OpenIMPACT. .
- **Nacho Navarro** is expert in Linux kernel and libraries
  - He is responsible for addressing OS-specific challenges in compiling, profiling, analyzing, and optimizing kernel code.
- **Robert Kidd** is a compiler engineer under GELATO support
  - He is responsible for scripting drivers as well as compiler-internal enhancements for increased GCC compatibility and compiler usability in general.
- **Shailesh Patel** is a compiler engineer
  - He is focusing on adding C++ support to the compiler.
- **Erik Nystrom** is interprocedural analysis architect
  - He is developing new scalable deep program analysis algorithms, innovative pointer analysis techniques in particular, to power future OpenIMPACT tools

# Today's Features

---

- Moving OpenIMPACT From Research To General Use
  - Robert Kidd
- C++ Support Status and Plans
  - Shailesh Patel
- OpenIMPACT ILP
  - Work of John Sias
- Pointer Analysis Overview
  - Work of Erik Nystrom
- Wrap-up
  - Wen-mei Hwu

# Moving IMPACT From Research To General Use

Robert Kidd

University of Illinois at Urbana-Champaign  
<http://www.gelato.uiuc.edu>

# OpenIMPACT

---

- ⊙ Why has it taken so long?
- ⊙ How'd you do it?
- ⊙ What's left?
- ⊙ When can I get it?

# Why has it taken so long?

---

- ◉ IMPACT originally a research compiler
- ◉ Reliance on proprietary code
  - New code scheduling library removes this obstacle
- ◉ Emphasis on research evident throughout compiler
  - Build system and installer
  - User interface
  - Compiler code library
  - ABI compatibility
  - Testing

# Research Compiler Advantages

---

- Cross file, profile driven inlining
- Profile driven optimization
- Interprocedural pointer analysis
  
- Fitting these into a normal Makefile requires significant work

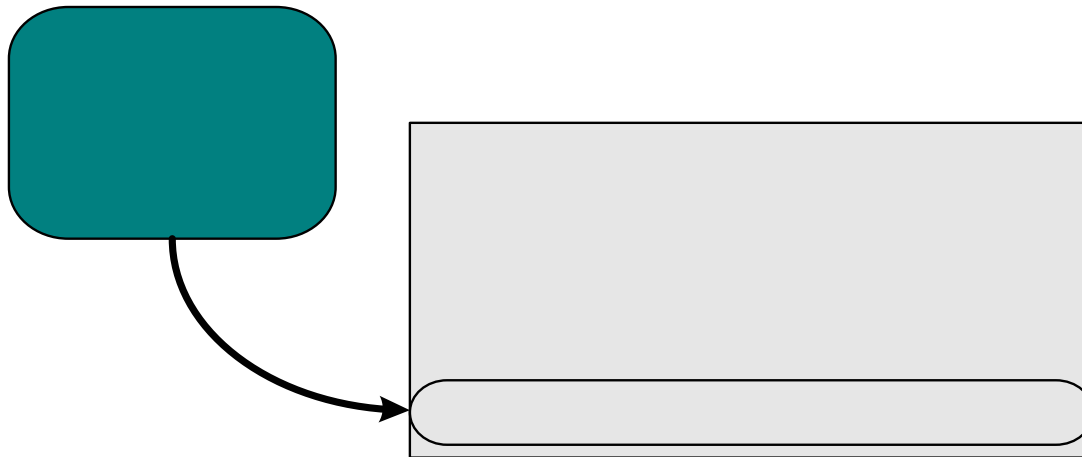
# User interface challenges

---

- IMPACT's advanced technologies require access to all source files
- It is only at link time that a gcc-like compiler has all files available
- If OpenIMPACT is going to fit gcc usage, will have to compile at link time

# User interface challenges

- Build systems can do more than compilation before link time
  - OpenSSL builds several libraries, then links into final program
- OpenIMPACT must discover source code during compile time, then access it at link time



# OpenIMPACT Strategy

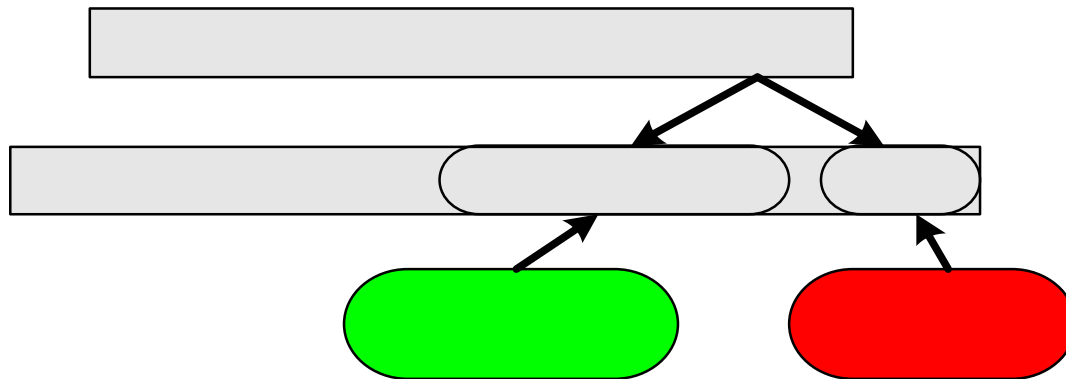
- OpenIMPACT embeds IR in object files generated during build system's compile stage
  - IR naturally survives object processing (copying, library building, ...)
  - IR can be extracted by OpenIMPACT for further processing
- Nice bonus: Since IR survives into libraries, we can now push optimization into appropriately prepared system libraries
- OpenIMPACT can now get all source files, so all is well
  - Not quite...

# UI is not enough



OpenIMPACT has access to all files and then some

- Libraries may contain unneeded files
- Libraries may be linked more than once
- Simple expansion of libraries wastes effort, duplicates symbols



# Pcode Linker

- Need to emulate Id to find set of IR to compile
- Need symbol table to emulate Id
  - Evolved into rewrite of front end library
- Benefits
  - Linker emulator
  - Better interprocedural analysis support
  - Supports an improved pointer analysis system

# oicc Goals

---

- ⊕ User cares about his program, not the compiler
- ⊕ We'd like a drop in replacement for gcc
  - make CC=oicc
  - While that may not be sufficient to get most benefit, further input modifications should be minimal
- ⊕ Also need to be able to plug into Eclipse

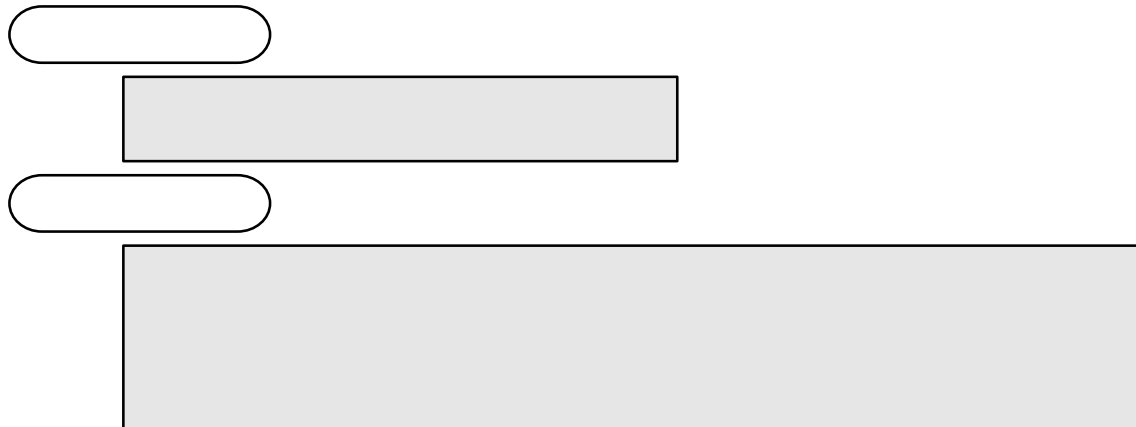
# So how do I use this thing?

---

- Change gcc to oicc, recompile
- What do I get?
  - Pointer analysis
  - Cross file inlining
- What do I miss?
  - Profile guided optimization
- PGO drives much of IMPACT's optimizations; as a result, small to moderate performance benefit

# What's necessary for PGO, then?

- Add a profiling step to build system
  - Link once
  - Execute profiling run
  - Link again
- Not out of line with other compilers



# Plink

- Id's symbol resolution is trivial
  - As long as symbol is expected type (func, var), linking successful
  - Linking hides a multitude of sins
- Plink attempts to create valid program from objects
  - Resolves symbols like Id
  - Merges types and structures
  - Ensures function calls make sense

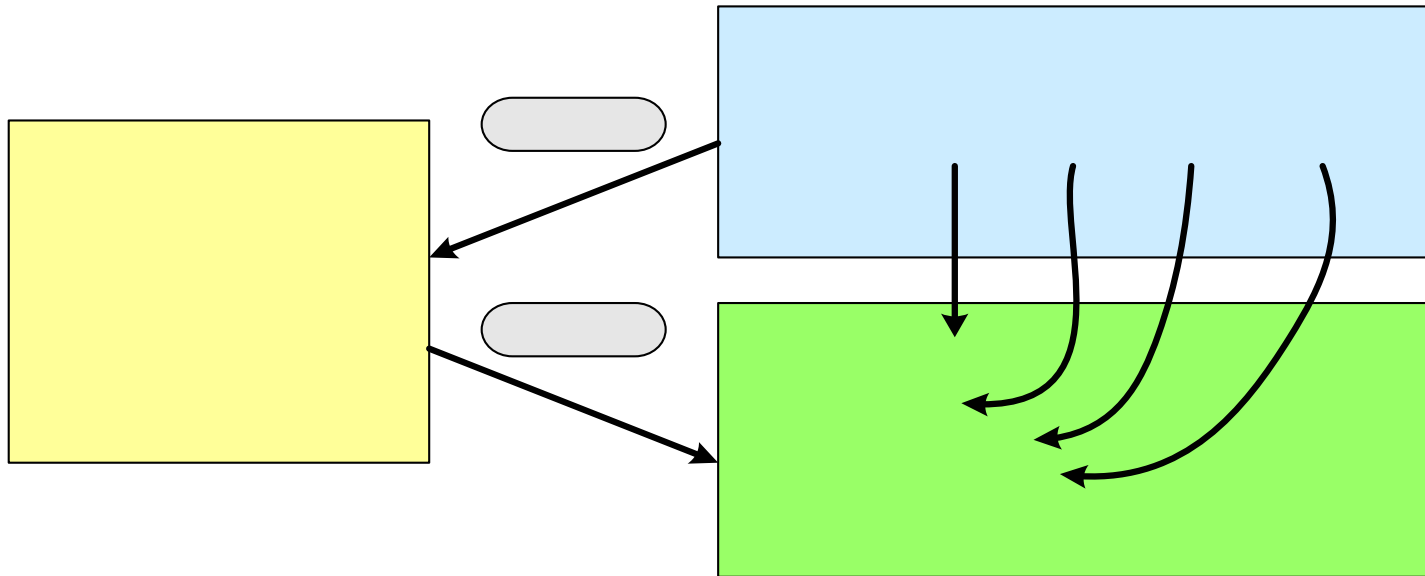
# Function merging

---

- Id does not check that a function is passed appropriate arguments
- Pcode naturally builds function prototypes
  - Calling function incorrectly will cause problems
- Necessary to clean up function calls

# Function call cleanup

- Execute arguments to preserve side effects
- Correct function call to make valid program



# What's left?

---

- Front end library
  - Inliner
  - Cleanup
  - Testing
- Interface
  
- Testing framework

# When can I get it?

---



Beta program starts on September 1

- Ask for code to compile with OpenIMPACT
- Ensure compiler can compile code
- Compile code, package, return
- Include copy of OpenIMPACT with compiled code.

# C++ Support Status and Plans

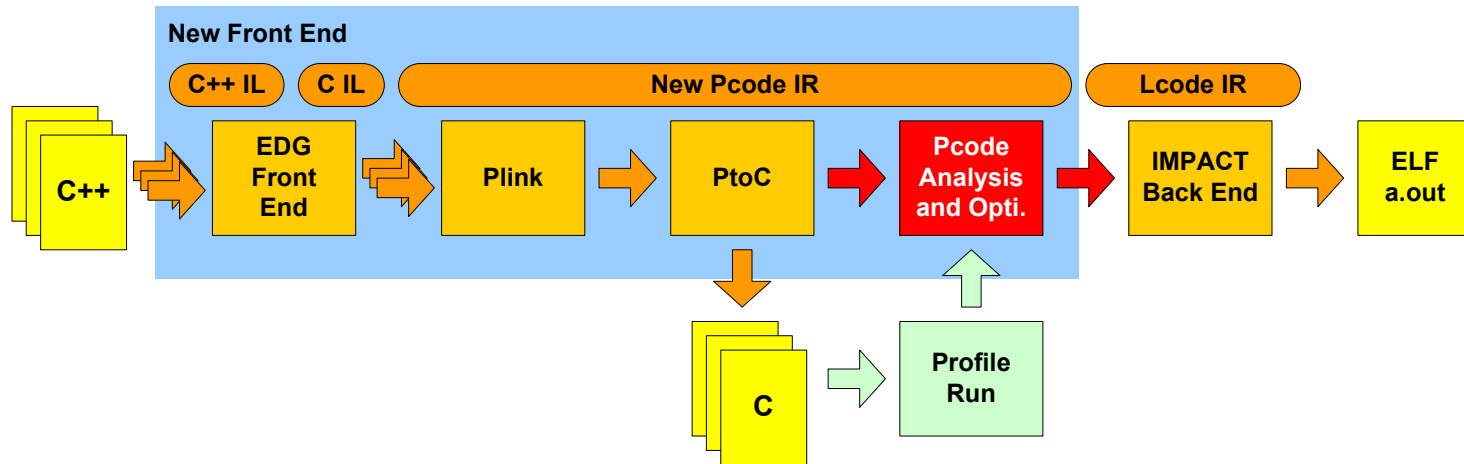
Shailesh Patel

University of Illinois at Urbana-Champaign  
<http://www.gelato.uiuc.edu>

# Developing OpenIMPACT C++ support

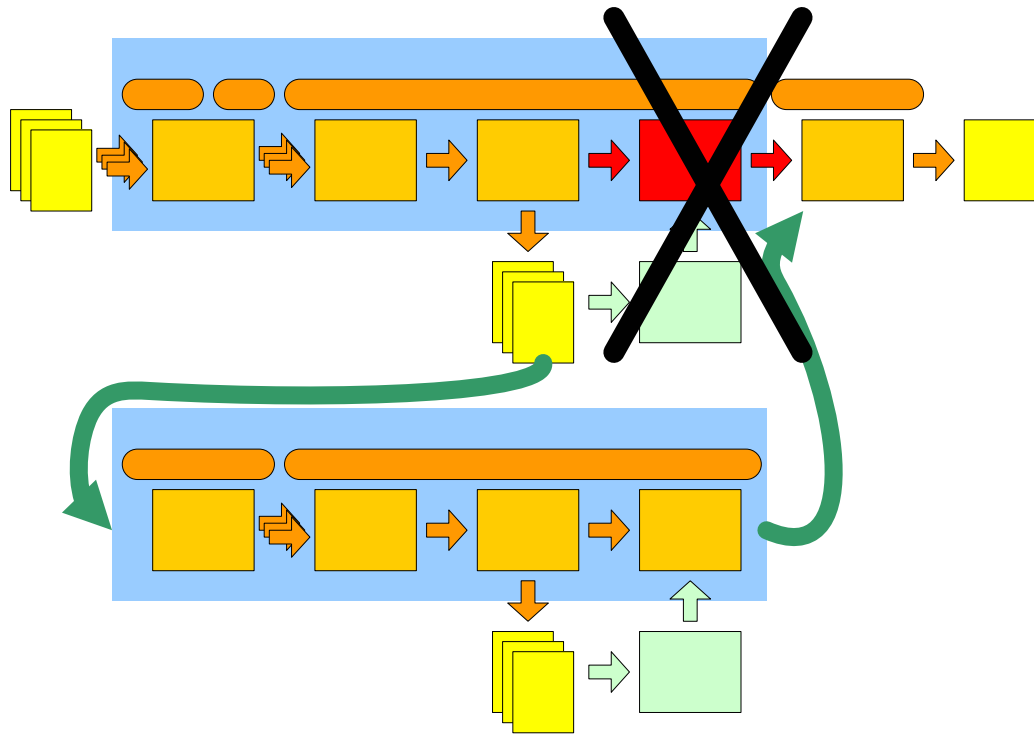
- IMPACT is traditionally a C compiler
- EDG today supports ANSI C++
- High demand for C++ support
  - SPECcpu2000 C++ applications (*252.eon*)
  - Gelato partner code (*ROOT, SpecNAMD,...*)
- *IMPACT's cross-module analysis and optimization support will benefit typical C++ apps*
- On step two of a three-step process
  - Upgrade of OpenIMPACT to latest EDG (**DONE**)
  - EDG-based *cf*ront-style support in new front end
  - full-fledged support, with C++-targeted optimizations

# *Cfront*-style C++ in new front end



- New OpenIMPACT front end uses EDG to translate C++ to C (like old *cfront* approach) and then to Pcode
- Plink stage provides link emulation not available in old front end, enabling complex source builds
- *But...* new front end is not yet complete
- Need testbed for prioritizing C++ work items

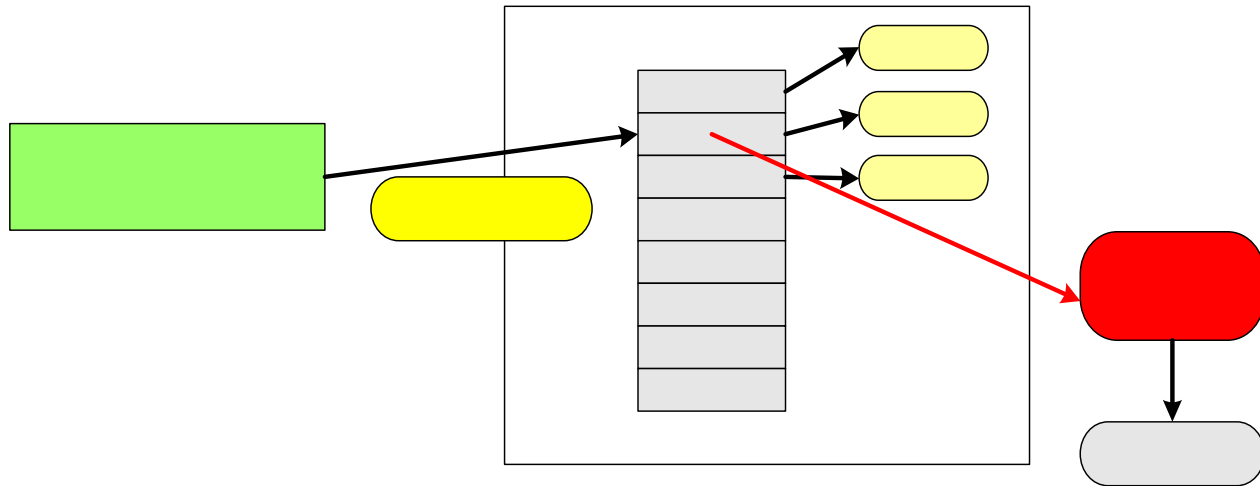
# Temporary C++ testbed solution



- Feed reverse-generated C from new front end into old C front end—works for *252.eon*

Ne

# Testbed results from *252.eon*

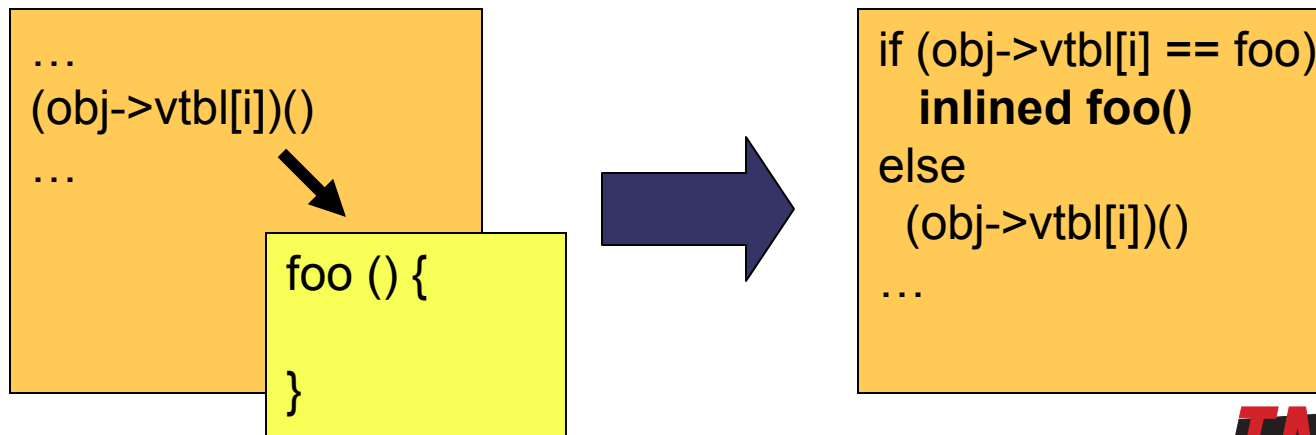


## C++ → C conversion loses important semantics

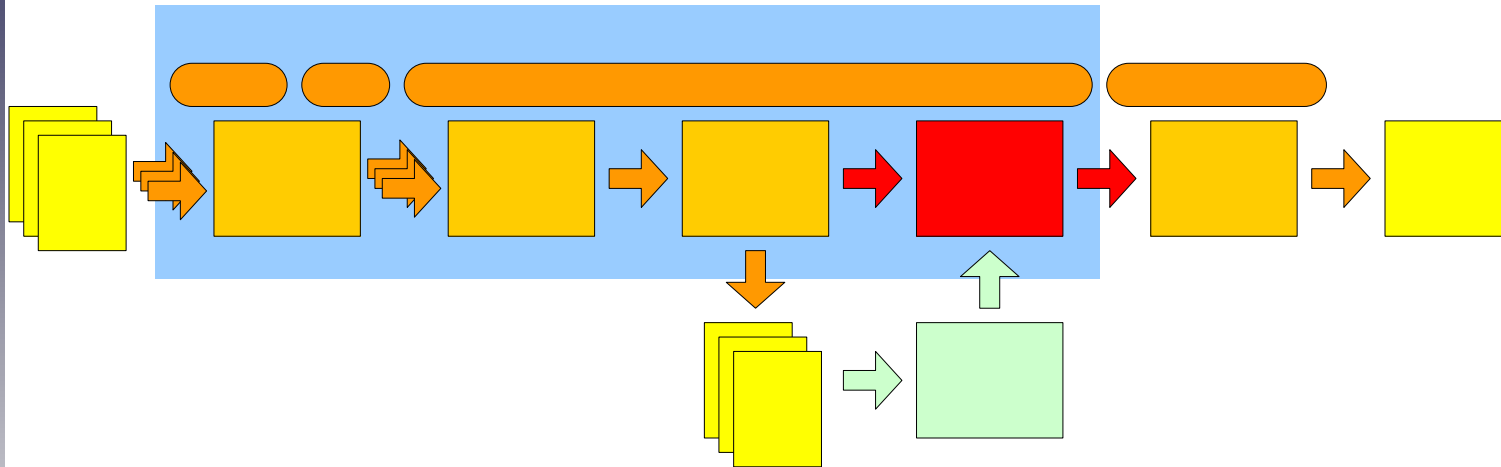
- Virtual function invocations
  - turn into indirect calls through tables of function pointers (*in EDG stored as long ints!*)
  - this violates assumptions made in old pointer analysis, so it must be disabled
- Class hierarchy / type information is obscured in resulting C
  - limits ability to perform C++-directed optimizations, including devirtualization of monomorphic virtual invocations

# Testbed results from *252.eon*

- ❖ Lack of pointer analysis a serious limitation
  - New pointer analysis coming online soon
  - Primary reason for mediocre performance ( GCC << OpenIMPACT << Intel ECC)
- ❖ Some C++ optimizations can be “faked”
  - > 95% of calls are inlined in *252.eon* using profile-guided conditional inlining; doubles performance
  - some overhead remains relative to devirtualization



# Finishing the Open IMPACT C++ path



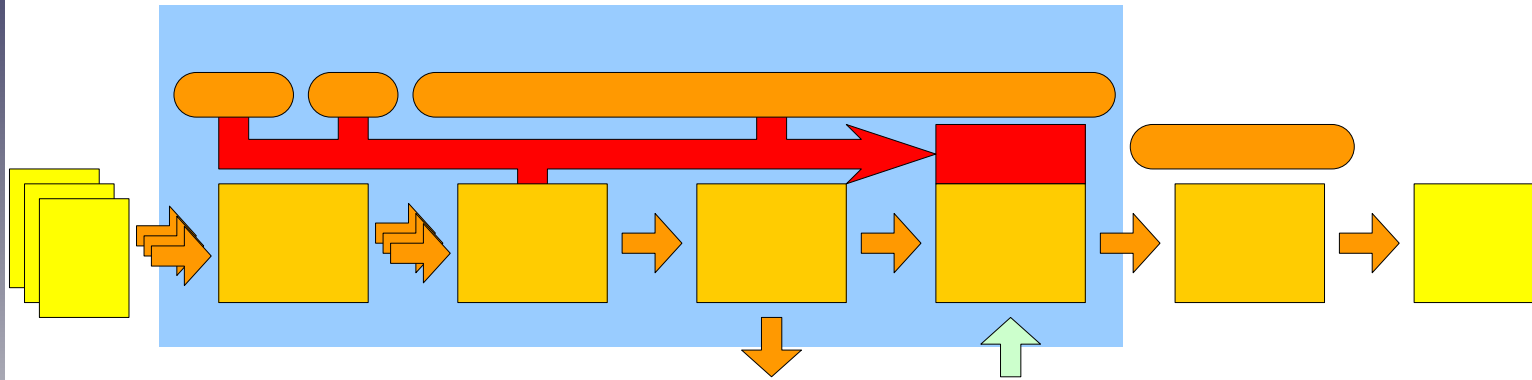
- Finish new front end (getting close)
  - Will enable pointer analysis for C++ apps
  - Retire “switchback” thru-C compilation path

**New Front End**

- Exercise and enhance *cfront*-like path
  - Expand C++ compiled code base
  - Prioritize C++-specific optimization needs

**C++ IL      C IL**

# Finishing the Open IMPACT C++ path



- Plan development of full-fledged C++ front end
  - Preservation of C++ information into Pcode
  - Implementation of C++ optimizations
- Need C++ “applications of interest” **New Front End (w**
  - Already investigating Gelato member’s code
  - Soliciting suggestions of other programs **C++ IL** **C IL** features to support.

# OpenIMPACT ILP

Work of John Sias

University of Illinois at Urbana-Champaign  
<http://www.gelato.uiuc.edu>

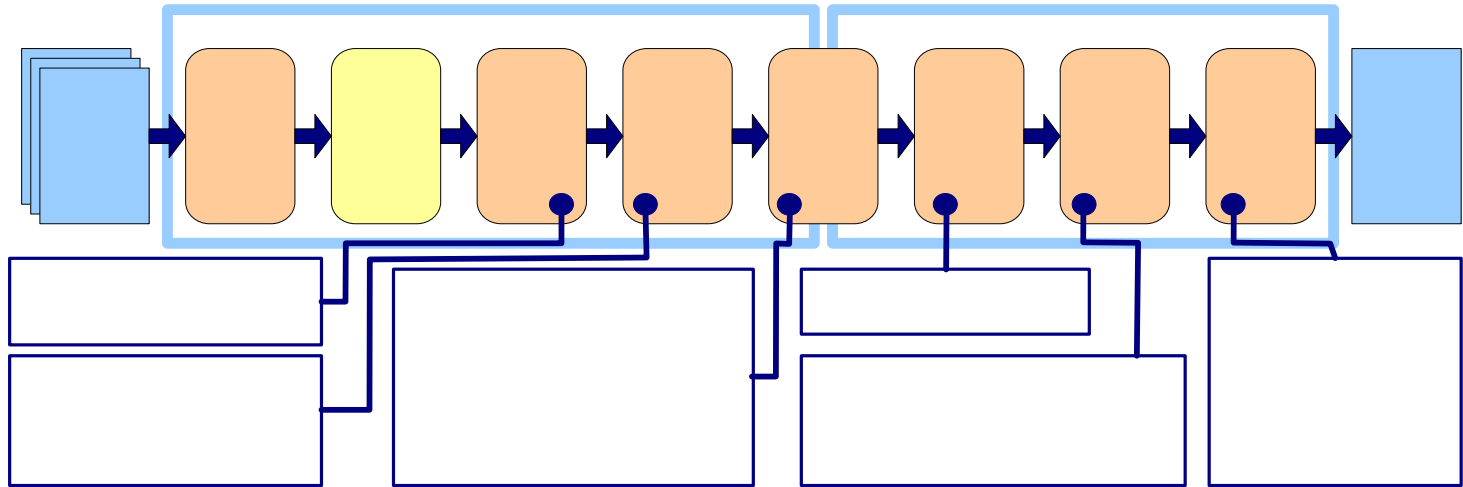
# OpenIMPACT: Theory and Practice

- Ongoing research in compilation for instruction-level parallelism
  - More accurate pointer analysis
  - More aggressive ILP-exposing transforms
  - Deeper understanding of compiled code base
  - *Findings improve IMPACT and other compilers*
- Simultaneous adaptation to more generally applicable form
  - More scalable pointer analysis
  - More stable ILP-exposing transforms
  - Larger compiled code base
  - *Benefits available to many applications in IMPACT*
- *This synergy is unique to OpenIMPACT*

# Getting inside the ILP lockbox

- ILP compilation breaks barriers to parallelism
- **Control** (branches/subroutine calls) breaks up instruction issue and optimization
  - Static (plan-time) problem (segmentation)
  - Dynamic (execute-time) problem (mispredicts)
- Need to discriminate between true and **false dependences**
- May need to mitigate “**occasionally true**” dependences
- May need to build in **tolerance for variable latencies** (e.g. data cache)

# IMPACT's "structural" approach



Applies EPIC techniques in a *structural simplification* paradigm (radical program changes)

- Profile-driven cross-file inlining, optimization
- Aggressive predication, control speculation in large regions
- At each step, enable as much future transformation as possible, but need to limit cost
- Arbitrary decisions driven with partial information – entails risk!

C  
source  
code

Pcode  
Pcode  
generation  
IMPACT  
ADVANCED COMPILER TECHNOLOGY

# Control optimization using EPIC



Typical: “incremental” (conservative)

- *Improve traditional global scheduling*
- May fill gaps in schedule, but doesn't fundamentally change the problem
- Unlikely to have substantial +/-



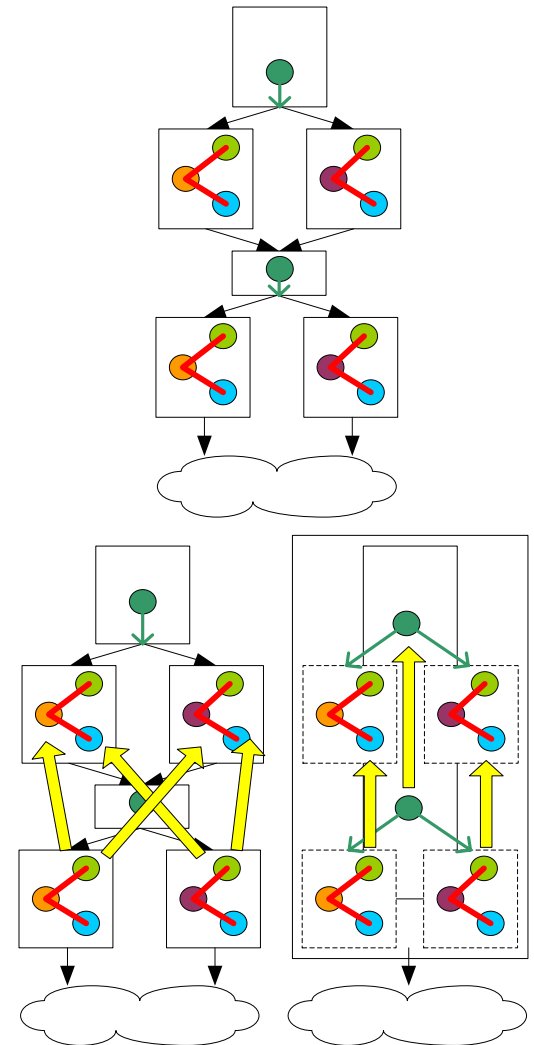
IMPACT: “structural” (radical)

- *Restructure program control flow*
- Form large, flexible execution regions
- Forming regions has costs → set “loss limits”

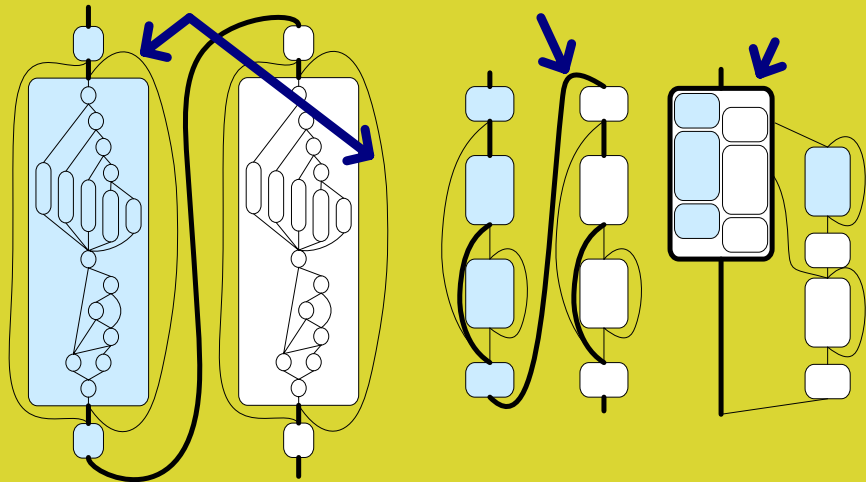


Our experience:

- many gains from radical approach, even relative to good incremental approaches
- setting current “loss limits” to screen all losses forfeits most benefit
- *Current applicability and future promise*



# Advanced ILP Transformations



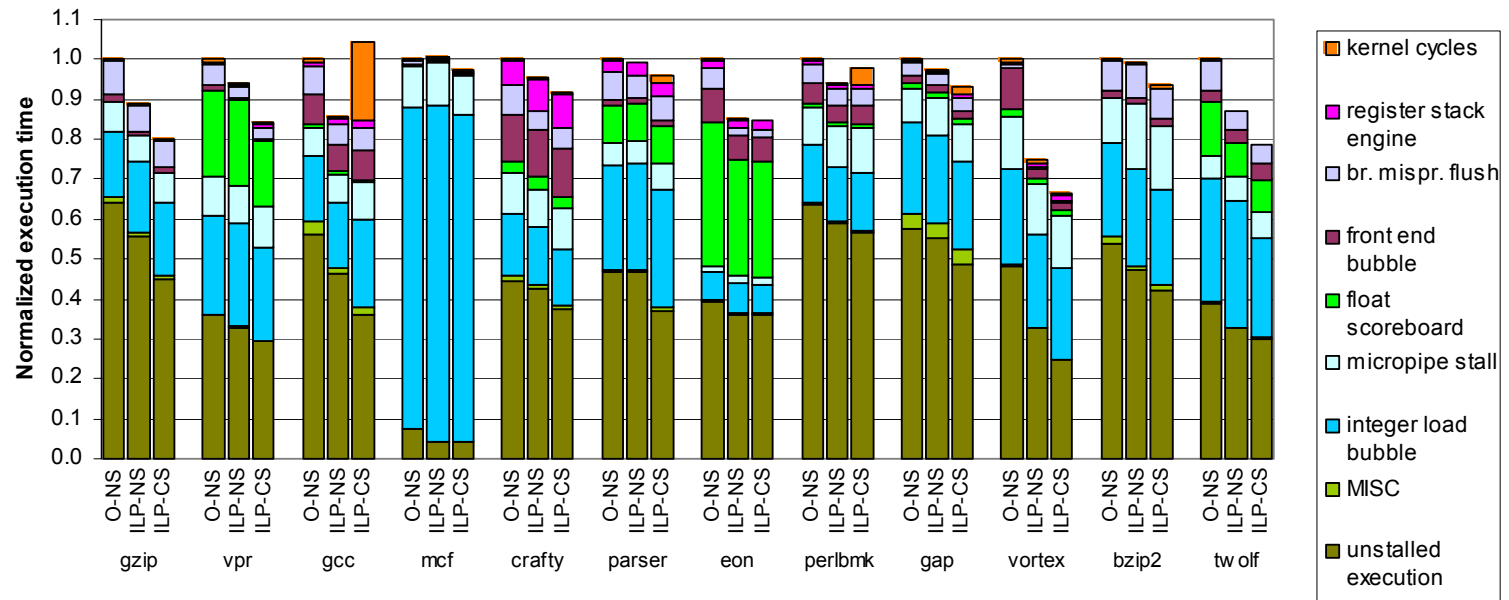
Example from SPECint2000 *crafty*

- Programmed structure limits ILP
  - low-tripcount ( $n \approx 1$ ) loops
  - internally serial loop bodies
  - frequent branching control
- Profile-based region formation radically transforms program to expose ILP
  - Eliminates internal control flow using predication
  - Peels loops to expose potentially-parallel iterations for interaction
  - Fuses compatible regions for inter-scheduling (using control speculation)

[SiasISCA04]

- OpenIMPACT's *structural transformation* techniques make aggressive frequently taken C-backed edges features

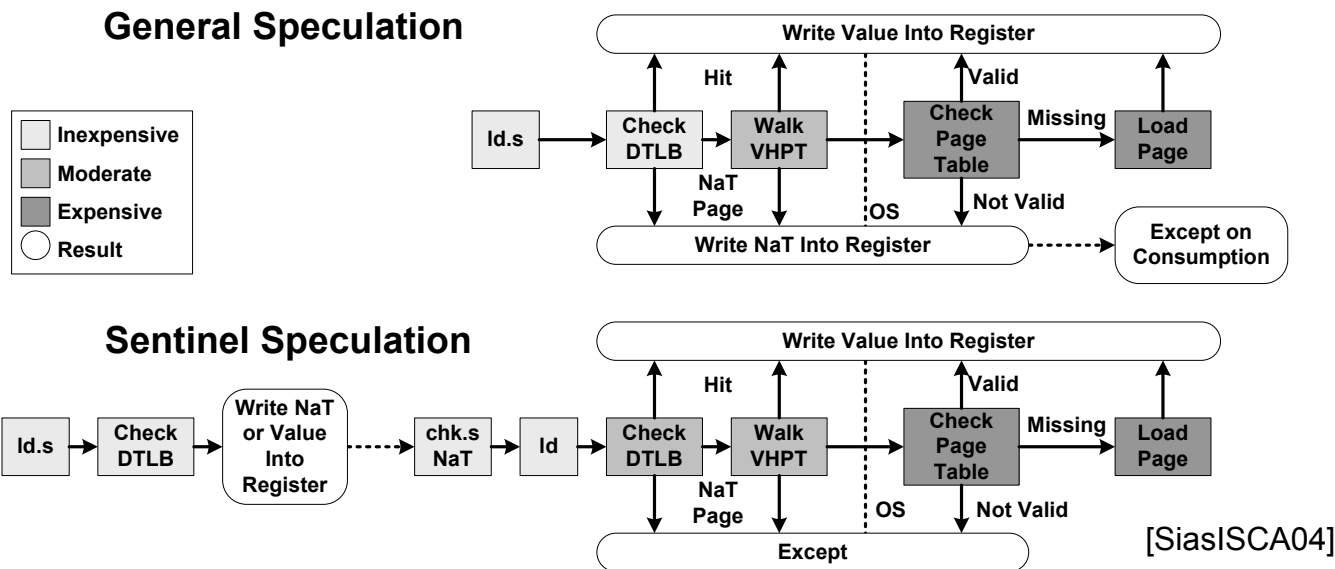
# Effects of ILP Transformations



HP zx6000 1GHz/3MB Itanium 2 [Sias|SCA04]

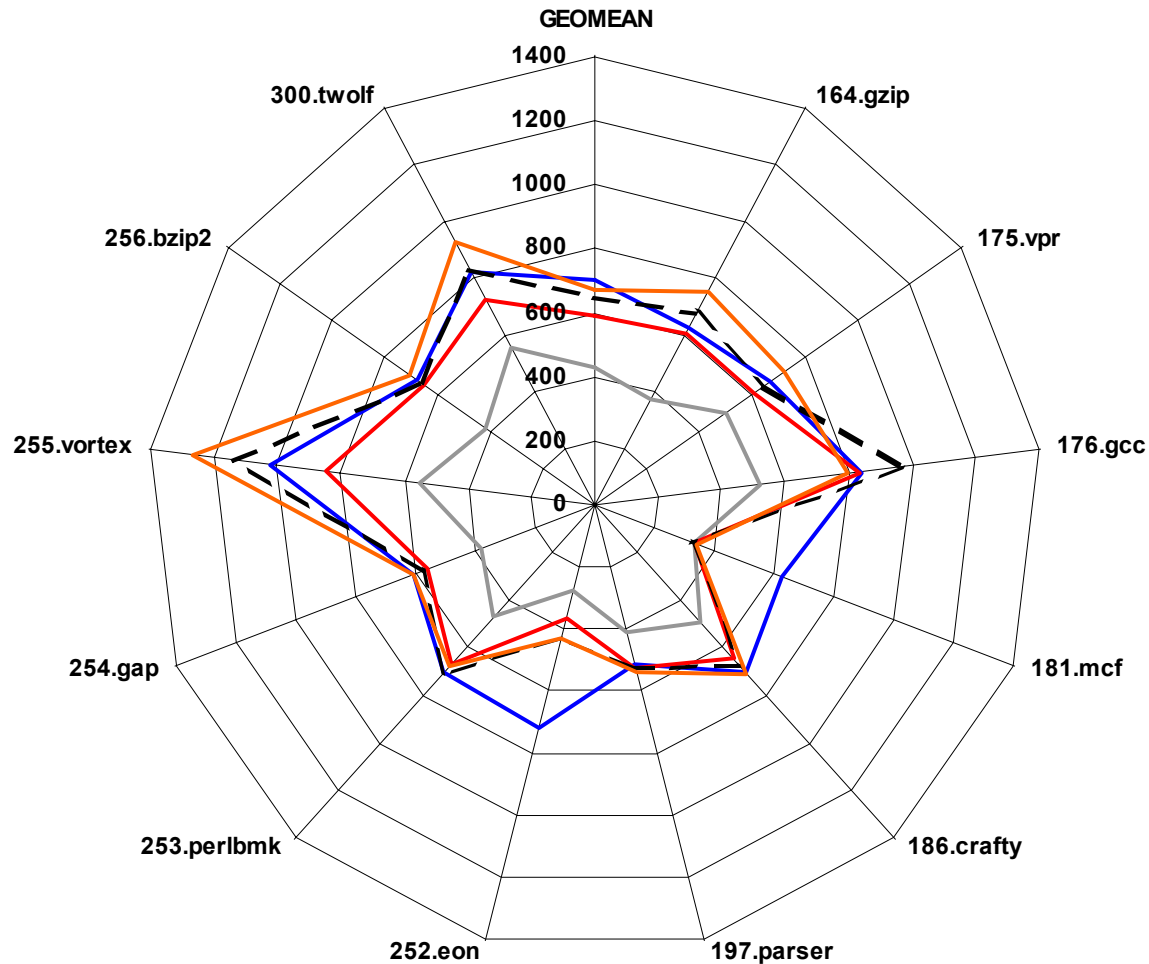
- Classical optimization (**O-NS**), ILP (**ILP-NS**), and ILP w/ control speculation (**ILP-CS**) configurations of OpenIMPACT. ILP features:
  - increase “planned ILP”
  - reduce branch misprediction time
  - generally do not degrade data cache performance
  - generally positively affect instruction cache performance.
- Increase in kernel cycles due to general speculation model

# General Speculation Model



- OpenIMPACT uses a modified linux kernel to enable control speculation without recovery code
  - Improved efficiency in many cases
  - Sometimes “wild loads” cause performance-degrading spurious page faults
- Wild loads often occur in specific cases that can be detected
  - The OpenIMPACT team is developing new analysis techniques to manage these loads

# Performance comparison



# Current ILP Projects / Challenges

- New, more comprehensive ILP optimization framework in development
  - Allow more aggressive transforms with better management of consequences
- New front end and memory dependence code
  - Increased opportunity for interprocedural optimization and global strategies
  - IPA will soon be available across SPECint2000
- Data speculation for instruction scheduling
  - Initial implementation ready, showing promise
- Control speculation model
  - Improving performance stability, comparing against sentinel model

# Pointer Analysis Overview

Work of Erik Nystrom

University of Illinois at Urbana-Champaign  
<http://www.gelato.uiuc.edu>

# Pointer Analysis

---

- Goal: describe potential pointer targets of each program variable
- Active area of research
- Immediately applicable to compiler
  - Load/store optimizations
  - Scheduling

# Pointer Analysis: Trade-off



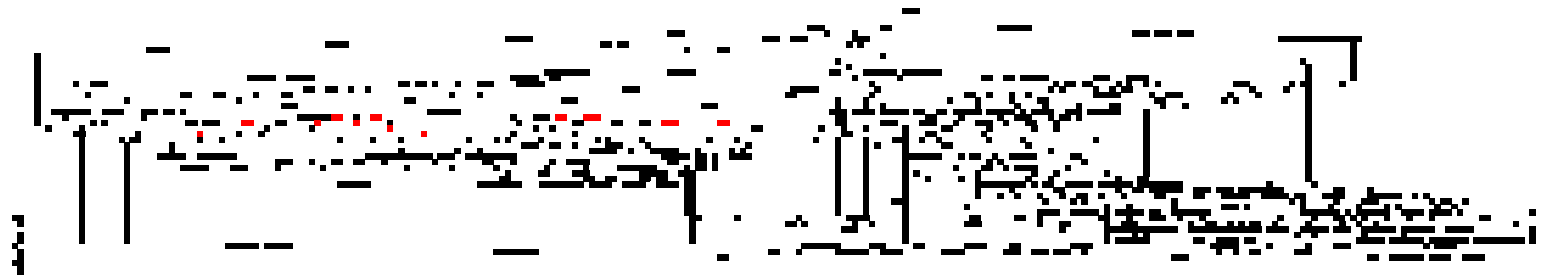
Historically, trade-off between accuracy and scalability

- Commercial implementations tend to be conservative to keep run time reasonable
- IMPACT's aggressive analysis enables better optimization, but this comes at a significant cost

# Pointer Analysis: Payoff



- Averages 10-30 targets per indirect call



- Averages 1.2 targets per indirect call

# Pointer Analysis: Issues



Current pointer analysis system lacks predictability

- Takes seconds for some applications
- Others run indefinitely
  - 253.perlbnk
  - OpenSSL
- Run time for analysis determined through trial and error

# Pointer Analysis: Motivation

---

- ⊕ Users demand keeping analysis cost low
- ⊕ Advanced optimizations demand keeping accuracy high
- ⊕ Dynamically adjust algorithm to maximize accuracy where it will not hurt performance

# Pointer Analysis: Goal

---

- Keep efficiency and accuracy without breaking safety
- Expand range of programs that can be processed
- One design point for all applications

# Pointer Analysis: Method

---

- Significant performance gain from rewrite
  - More efficient implementation
- Analysis system can regulate amount of accuracy to attempt
  - Regulation on per-variable level
- Regulation allows flexibility that didn't exist before
  - Pointer analysis is no longer all or nothing

# Performance results

Benchmark	PIPAnal	Context Insensitive	New analysis
008.espresso	9	2	6
130.li	1332	9	2
132.ijpeg	85	4	2
134.perl	408	3	14
176.gcc	Hours	45	38
253.perlbnk	Months	1000	530
254.gap	3350	120	55
255.vortex	136	6	20
300.twolf	2	1	1

# Pointer Analysis

- More general pointer analysis is important for OpenIMPACT
  - Need a pointer analysis system that can easily process any input program
- New flexibility is a big win
  - Pointer analysis benefits are currently all or nothing
- Many optimizations depend on analysis
  - 'Nothing' is very unattractive

# Acknowledgements

- Former IMPACT members
  - David August, Ben-Chung Cheng, Daniel Connors, Kevin Crozier, Brian Deitrich, John Gyllenhaal, Richard Hank, Teresa Johnson, Dan Lavery, Scott Mahlke, Le-Chun Wu
- Intel Itanium Team
  - Carole Dulong, John Crawford, Dan Lavery, Steve Skedzielewski, Jim Pierce
- HP Itanium Team
  - Richard Holman, Vatsa Santhanam, Carol Thompson, Richard Hank
- HP Labs CAR Team
  - Bob Rau, Mike Schlansker, Vinod Kathail, Scott Mahlke
- HP Labs Linux Team
  - Brian Lynn, David Mosberger, Hans Boehm, Stephane Eranian
- HP Philanthropy – Itanium and Itanium 2 grants
  - Karen Fontana, Tony Napolitan, Ralph Hyver, Chris Hsiung, Rob Bouzon, Gregg Peters, Rob Reed
- Intel - 2 alpha/beta Itaniums
  - Carole Dulong, Richard Wirt