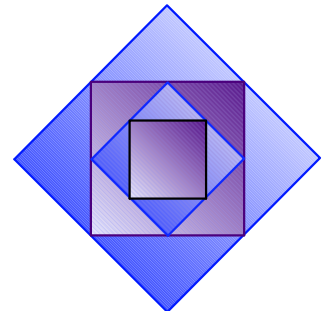


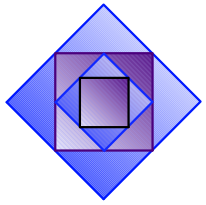
Post Link Optimization

Ealan Henis
Gadi Haber
Moshe Klausner
Alex Warshavsky

November 1999

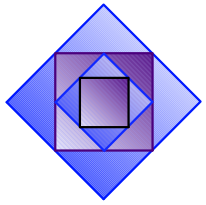
IBM Research Lab in Haifa





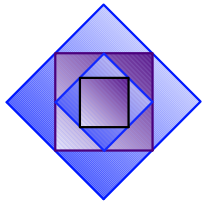
Plan of the talk

- **Introduction**
- **Methods of analysis**
- **Results**
- **Conclusions**



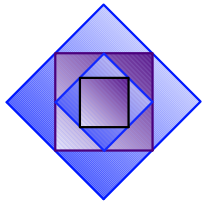
Introduction

- **Processors are faster than memories**
- **Faster memory access via**
 - Hardware: cache hierarchies, prefetch buffers
 - Software: branch prediction, Translation Lookaside Buffers
- **Cache and branch prediction misses result in performance penalty**



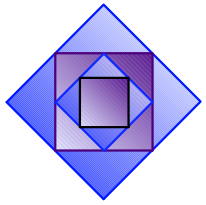
Compiler optimizations

- **Compilers optimize for speed, space etc.**
- **Non feedback analysis:**
 - paths are equally likely
- **Feedback analysis:**
 - limited to compile unit scope



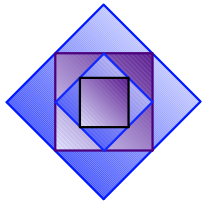
Code repositioning

- **Increased code/data locality**
- **I-cache, TLB, page faults, branch penalty**
- **Units: procedure, superblock, basic block**
- **Profiles**
 - paths
 - CFG edges and nodes
- **References:**
 - Pettis & Hansen 90; Gloy et al. 97
 - Ball and Larus 96
 - McFarling 91, Young 98 theses
 - Heisch 94; Nahshon & Bernstein 96



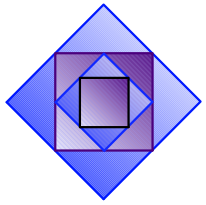
Our optimization approach

- **Post link analysis: global optimizations**
- **Break compiler and linker conventions**
- **No source code required**
- **Optimize linked-in library code as well**
- **Shared libraries, multiprocesses**
- **Scaleability**
- **Current tool versions:**
 - PowerPC/AIX - product
 - x86/NT and OS/390 - research prototype



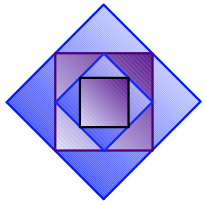
Methods of analysis

- **Program instrumentation**
- **Program profiling**
- **Optimization**



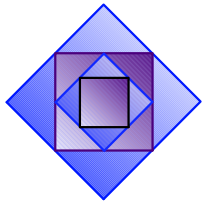
Incremental disassembly

- **Dissect program into basic blocks**
- **Imperative to cover the entire code**
- **Difficult: code and data are intermingled**
- **CISC instruction set**
- **Scaleability**
- **Approach**
 - Extract pointers to code
 - Identify and characterize basic block
 - Resolve all possible control flow paths
 - Mark areas not discovered as "unclassified"



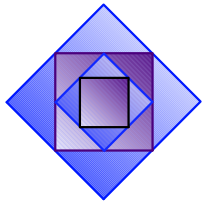
Program instrumentation

- **Generate a new executable for recording basic block utilization**
- **Maintain original semantics and flow**
- **Steps:**
 - Disassemble and identify basic blocks
 - Instrument basic blocks to track edges/nodes
 - Generate a new executable
- **Override vs. extend BBs, universal stub**



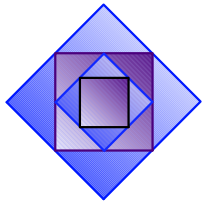
Program profiling

- **Generate usage statistics**
- **Select representative workloads**
- **Applicable to shared DLLs**
- **Accumulative profiling**



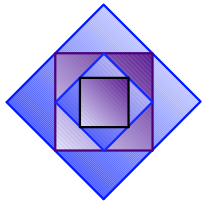
Optimization steps

- **Maintain original semantics**
- **Disassemble into basic blocks**
- **Read-in feedback information**
- **Analyze and optimize**
- **Generate a new executable**



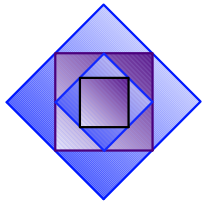
Optimizations

- **Code repositioning**
 - Heatshrink vs. AOPT
- **Data compaction, padding elimination**
- **Alignment**
- **Conditional branch reversal**



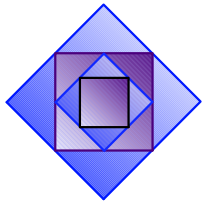
Optimizations (AIX)

- **TOC dereferencing**
- **Elimination of save/restore for unused registers**
- **Selective function inlining**
- **Basic blocks inlining**
- **Insert instructions between compare and branch**



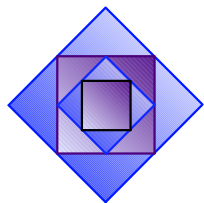
Generation of a new executable

- **Fix modified addresses**
- **Trampolines**
- **Long vs. short jumps**
- **Detailed rebuild of some sections**
 - relocations (NT)
 - DLL exports (NT)
- **Traceback tables (AIX)**
- **Debug symbols**



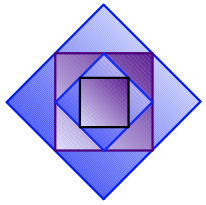
Results

- **Incremental disassembly succeeds**
- **Method is scaleable**
- **Analysis time**
 - 20 minutes for 50 MByte DLL (x86/NT)



Results

platform	program name	program size	workload	instrumented run overhead (%)	optimized run gain (%)	comments
AIX	compress	470K	ref	220	3	of SPEC95
AIX	li	619K	ref	636	13	of SPEC95
AIX	app-D	28000K	workload-D	600	23	server app, internal use
NT	app-A	390K	workload-A	980	3	app for internal use
NT	app-B	63000K	workload-B	280	4	server app, internal use
NT	app-C	13700K	workload-B	240	3	server app internal use
OS/390	ijpeg	401K	penguin	400	2	SPEC95
OS/390	compress	82K	bigtest	580	2	SPEC95
OS/390	m88ksim	467K	ctl	800	10	SPEC95



Conclusions

- **Compilers cannot do post link optimizations**
- **Complement compilers:**
 - Improve performance on top of compiler optimizations
- **Optimize library code as well**
- **Scaleable, Robust:**
 - suitable for large programs
- **May serve as a basis for future tools**