Dynamic Compilation

Advanced Topics:

- Dynamic compilation

  Today:

  - Finish How to Compile
  - Inlining in ahead of time and dynamic compilers

- Garbage collection
- Alias Analysis
- Interprocedural Analysis
- Dependence Testing
- Locality and parallelization

Inlining

Benefits and costs

- Direct benefit: reduced call overhead
  - allocating and deallocating local variables
  - saving and restoring registers *call convention*
  - pushing and popping parameters on to the stack
- Indirect benefit:
  - exposes optimization opportunities
- Direct cost:
  - code size increases (mostly) and can thus can degrade instruction cache behavior
- Indirect/direct cost:
  - increases compile time and space, e.g., tips the register allocator to spill

Inlining

Results vary across language and compiler

J. W. Davidson & A. M. Holler, "A Study of a C Function Inliner", *Software—Practice and Experience*, 18:8, August 1988

K.D. Cooper, M. Hall, & L. Torczon, "An Experiment With Inline Substitution," *Software–Practice and Experience*, 21:6, June 1991

K.D. Cooper, M. Hall, & L. Torczon, "Unexpected Side Effects of Inline Substitution: A Case Study," *ACM LOPLAS* 1(1), March 1992

M. D. Bond & K. S. McKinley, "Practical Path Profiling for Dynamic Optimizers," CGO, March 2005.

K. Hazelwood and D. Grove, "Adaptive Online Context-Sensitive Inlining" *International Symposium on Compiler Code Generation and Optimizations* (CGO), pp. 253–264, San Francisco, CA March 2003.

Profitability of Inlining

Davidson & Holler

- non-optimizing C compilers
- only 1 had register allocation

Cooper, Hall, & Torczon

- highly optimizing Fortran compilers
- compile times increased, some dramatically

Using simple heuristics to guide inlining, program growth not a problem in either study

average	DH	CHT
|lines|	1725 C	3103 Fortran
range of lines	217 - 6781	321 - 5979
|procedures|	50	27
original proc length		66
inlined proc length		297
static calls	157*	83
dynamic calls	3,123,118	
% statically inlined		75 %
% dynamically inlined		89 %

$*$: static calls exercised dynamically

Profitability of Inlining

Davidson & Holler

Improvement due to inlining

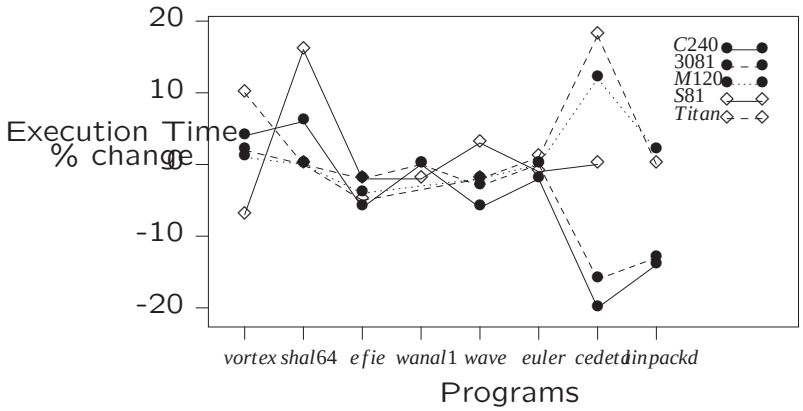
(with no optimizations, register displacement)

processors	inlining strategies intra-module: program in 1 module	intra-module: > 1 module	inter module
VAX-8600	15%	7%	8%
MC68020	6%	4%	5%
Clipper	7%	1%	0%
Convex	11%	4%	2%

Profitability of Inlining

Cooper, Hall, Torczon

Improvement/degradation due to inlining and optimizations



Secondary effects of inlining hard to predict

- eliminated 89% of dynamic calls on average $\Rightarrow$ (5 were $>$ 99.5%)
- eliminated 75% of static calls on average
- good compilers
  (Convex, IBM, MIPS, Stellar, Ardent)

Another Data Point - Bond/McKinley CGO

C & Fortran SPEC 2000 programs; Scale Compiler

	% calls inlined	Avg. unroll factor	Speedup
vpr	71%	1.65	0.97
mcf	98%	1.00	1.01
crafty*	0%	1.00	1.00
parser	29%	1.46	1.03
perlbmk*	14%	1.00	1.02
gap	59%	1.22	1.02
bzip2	49%	1.99	1.07
twolf	23%	2.19	0.96
INT Avg	43%	1.44	1.01
wupwise	0%	1.90	0.98
swim	0%	4.00	1.02
mgrid	10%	4.00	0.96
applu	0%	1.31	1.14
mesa*	0%	2.31	1.00
art	100%	4.00	1.06
equake	100%	2.97	1.03
ammp	98%	1.81	1.02
sixtrack	57%	3.35	1.29
apsi	100%	3.90	1.02
FP Avg	46%	2.96	1.05
Overall Avg	45%	2.28	1.03

Inlining and Unrolling.
*No cross-module inlining.

Adaptive Policies for Inlining

Hazelwood & Grove

- OO language encourages small methods
  - inlining ameliorates this cost
- JIT:
  - application exposed to compile time costs of inlining
- Static profiling (not compatible with dynamic class loading):
  - only inline hot methods
- Their policies:
  - Online profiling and inlining
  - Context sensitive profiling
  - Adaptive policies

What does context sensitivity mean?

insensitive sensitive

Remember the Jikes RVM JIT profiling structure?

Separation of dynamic profile gathering and policy

Sample time: gather profile information

- examine the stack
- identify hot method
- context insensitive: (caller,callee)
- context sensitive: (caller1,callsite1,...,callerN,callsiteN,callee)

  Policies for choosing N

  - Fix N
  - Dynamic: no parameters? N=1
  - Dynamic: Stop when the tracer sees a method call (N=2 typically)
  - Dynamic: Stop when the method is too large (N=4 typically)
  - Dynamic Hybrids
  - Unimplemented: exclude biased call sites, and make N large for hard to predict method invocations
- build the dynamic call graph annotated with edge frequencies
- decay old information

Inlining Heuristics

Jikes RVM Static Policies

- Always inline statically known tiny methods
  - Tiny = smaller than 2 times the size of the call instruction sequence (CIS)
- Subject to code growth budget and inlining depth bound
  - Inline small (2-5x CIS) statically known methods
  - Never Inline medium (5-25x CIS) methods
  - Never inline large methods (25x CIS or more)

Jikes RVM Dynamic Policies, when the controller reoptimizes a hot method

- Subject to code growth budget and inlining depth bound
  - Inline and guard small (2-5x CIS) methods
  - Inline (and if needed, guard) hot medium (5-25x CIS) methods using context sensitive call graph
- Inline hot small (2-5x CIS) methods even when code growth and inlining depth exceeded

Hazelwood & Grove: Bottom Line

Context sensitive inlining vs. static heuristics for inlining

- Inlining is needed in this setting
- Simple policies work, better policies reduce compile time, but don't help performance that much
- Average reduction in compile time and space: 10%
- Compile time reductions up to: 33% space, and 56% time
- Average code size reductions: approximately 10%
- Performance between -4% and 5%

Profitability of Inlining

Compilers are engineered objects

- implementations rely on properties of hand-written code
  finite limits on data structures
  assumptions get violated (*global regs*)
- call sites provide *big* hints
  clear out all registers
  limits "global" impact
- smaller procedures map better onto small resources
  32 (or 32 + 32) registers
  less control flow ⇒ better knowledge

Separation of concerns

- optimization ignores resource constraints
- global methods work well with plentiful resources
- bigger scope ⇒ separation can break down

Inlining is not a panacea

Next Time

Garbage collection

Read: Blackburn, Cheng, and McKinley, Myths and Realities: The Performance Impact of Garbage Collection, SIGMETRICS, pp. 25-36, New York, NY, June 2004.