Dynamic Compilation II: DyC and DELI

John Whaley, Janani Ravi

EE392c Spring 2003
Lecture 14

May 15, 2003

Agenda

- Overview of the two systems
- DyC
  - DyC System Overview
  - DyC's Run-Time Optimizations
  - Performance Analysis and Results
- DELI
  - DELI System Overview
  - DELI API
  - Example Application and Results
- Discussion

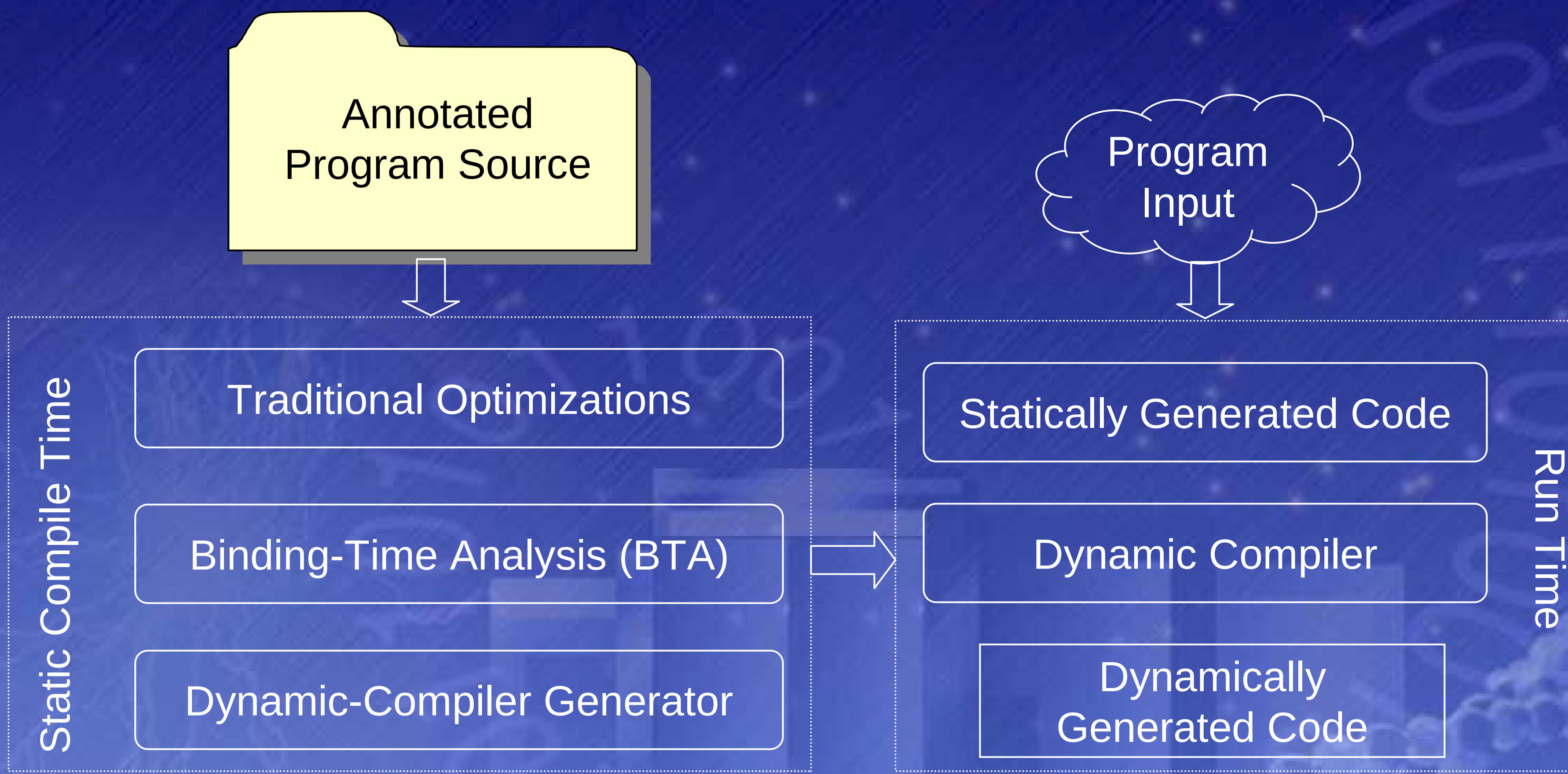
Overview

- Talk about today: two systems so that programmers can use dynamic compilation
- DyC: a version of C that includes an interface to a partial evaluator
  – Value-specific optimization of run-time constants
  – Help from static compiler: ‘staged’ compilation
- DELI: extension of Dynamo, includes an interface to manage code fragments
  – Explicit, fine-grained control over fragments
  – Simple dynamic compilation, useful for emulation

What is DyC?

- University of Washington, 1999
- Selective, Value-Specific Dynamic Compilation System
- Run-Time constants
- Targets complex C programs
- Declarative, annotation-based system
- Staged optimization
- Low overhead

DyC System Overview



Dynamic-to-Static Promotions and Polyvariant Specialization

- Specialization – generate multiple versions of code specialized to different values of static variables
- Static variables are said to be *promoted* from dynamic to static

Polyvariant Division

- The same program point may be analyzed multiple times
- Each time, a different set of variables is assumed static
- Programmer can annotate *conditional specialization*

Example

```
void do_convol (float image [][], int irows, int icols,
    float cmatrix [][], int crows, int ccols, float outbuf [][]) {
float x, sum, weighted_x, weight;
int crow, ccol, irow, icol, rowbase, colbase, crowso2, ccolso2;
make_static (cmatrix, crows, ccols, crow, ccol);
crowso2=crows/2; ccolso2=ccols/2;
for (irow=0; irow < irows; ++irow) {
    rowbase = irow-crowso2;
    for (icol=0; icol < icols; ++icol){
        colbase = icol-ccolso2; sum = 0.0;
        for (crow=0; crow<crows; ++crow) {
            for (ccol=0; ccol<ccols; ++ccol) {
                weight = cmatrix @[crow] @[ccol];
                x = image[rowbase+crow][colbase+ccol];
                weighted_x = x * weight; sum = sum + weighted_x; }}
        outbuf [irow][icol] = sum; }}}
```

Example – Partially Optimized

```
for (irow=0; irow < irows; ++irow) {
    rowbase = irow-1;
    for (icol=0; icol < icols; ++icol){
        colbase = icol-1; sum = 0.0;
        x = image[rowbase+0][colbase+0]; // Iteration 0: crow=0, ccol=0
        weighted_x = x * 0.0; sum = sum + weighted_x;
        x = image[rowbase+0][colbase+1]; // Iteration 1: crow=0, ccol=1
        weighted_x = x * 1.0; sum = sum + weighted_x;
        x = image[rowbase+0][colbase+2]; // Iteration 2: crow=0, ccol=2
        weighted_x = x * 0.0; sum = sum + weighted_x;
        x = image[rowbase+1][colbase+0]; // Iteration 3: crow=1, ccol=0
        weighted_x = x * 1.0; sum = sum + weighted_x;
        … //Iterations 4-8
        outbuf [irow][icol] = sum; }}}
```

Example – Partially Optimized

```
for (irow=0; irow < irows; ++irow) {
    rowbase = irow-1;
    for (icol=0; icol < icols; ++icol){
        colbase = icol-1;

        x = image[rowbase][colbase+1]; // Iteration 1: crow=0, ccol=1
        sum = x;
        x = image[rowbase+0][colbase+2]; // Iteration 2: crow=0, ccol=2
        weighted_x = x * 0.0; sum = sum + weighted_x;
        x = image[rowbase+1][colbase+0]; // Iteration 3: crow=1, ccol=0
        weighted_x = x * 1.0; sum = sum + weighted_x;
        … //Iterations 4-8
        outbuf [irow][icol] = sum; }}}
```

Example – Fully Optimized

```
for (irow=0; irow < irows; ++irow) {
    rowbase = irow-1;
    for (icol=0; icol < icols; ++icol){
        colbase = icol-1;
        // Iteration 0: crow=0, ccol=0
        //All code eliminated
        x = image[rowbase][colbase+1]; // Iteration 1: crow=0, ccol=1
        sum = x;
        // Iteration 2: crow=0, ccol=2
         //All code eliminated
        x = image[rowbase+1][colbase]; // Iteration 3: crow=1, ccol=0
        sum = sum + x;
        … //Iterations 4-8
        outbuf [irow][icol] = sum; }}}
```

Image Convolution

0	-1	0
-1	5	-1
0	-1	0

Sharpen

-1	0	0
0	0	0
0	0	-1

Emboss

0	1	0
1	0	1
0	1	0

Example

Performance Analysis

- Workload - Applications

  dinero – cache simulator

  m88ksim – Motorola 8800 simulator

  mipsi – MIPS R3000 simulator

  pnmconvol – image convolution

  viewperf - renderer

Performance Analysis

- Workload - Kernels

  binary – binary search over an array

  chebyshev – polynomial function approximation

  dotproduct – dot-product of two vectors

  query – tests database entry for match

  romberg – functional integration by iteration

Performance Analysis

- System
  - DEC Alpha 21164 based workstation, 1.5 GB RAM - lightly loaded
  - Multiflow compiler
    - comparable to gcc –O2

Performance Analysis – Static Variables

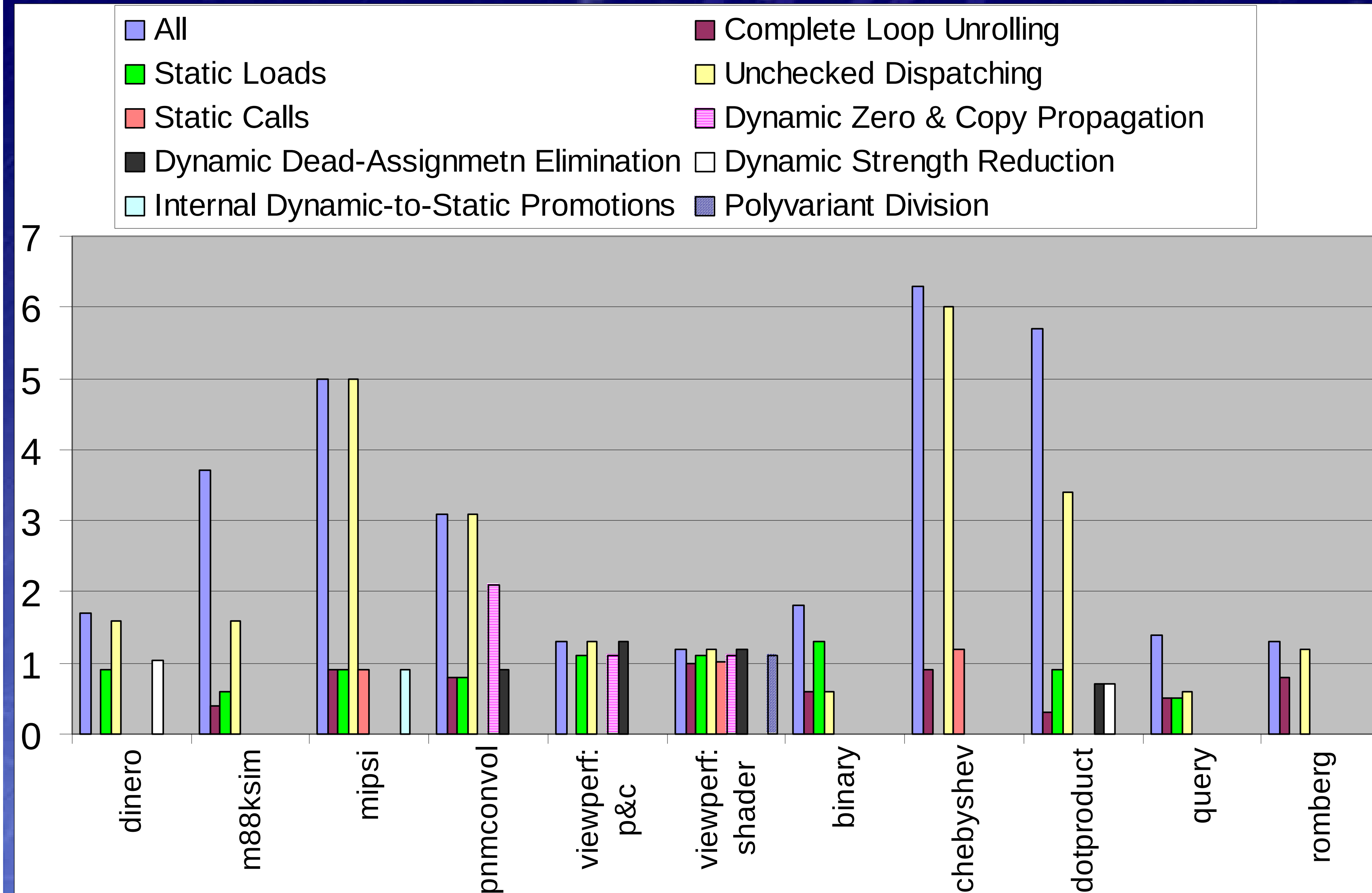
Program	Annotated Static Variables	Values of Static Variables
dinero	cache configuration	8kB I/D, direct-mapped, 32B blocks
m88ksim	an array of breakpoints	no breakpoints
mipsi	input program	bubble sort
pnmconvol	convolution matrix	11x11 with 9% ones, 83%zeros
viewperf	3D projection matrix, lighting vars	perspective matrix, one light source
binary	input array and its contents	16 integers
chebyshev	the degree of the polynomial	10
dotproduct	contents of one of the vectors	a 100-integer array with 90% zeros
query	a query	7 comparisons
romberg	the iteration bound	6

Measurement results

Program	Asymptotic Speedup	Break-Even Point	Overhead (CPI)	Instr. Generated
dinero	1.7	1 invocation(3524 memory references)	334	634
m88ksim	3.7	28 breakpoint checks	365	6
mipsi	5.0	1 invocation (484634 instructions)	207	36614
pnmconvol	3.1	1 invocation (59 pixels)	110	2394
viewperf:project&clip	1.3	16 invocations	823	122
viewperf: shade	1.2	16 invocations	524	618
binary	1.8	836 searches	72	304
chebyshev	6.3	2 interpolations	31	807
dotproduct	5.7	6 dot products	85	50
query	1.4	259 database entry comparisons	53	71
romberg	1.3	16 integrations	13	1206

Measurement results

Program	Asymptotic Speedup	Execution Time in the Dynamic Regions (% of total static execution)	Average Whole Program Speedup
dinero	1.7	49.9	1.5
m88ksim	3.7	9.8	1.05
mipsi	5.0	~ 100	4.6
pnmconvol	3.1	83.8	3.0
viewperf	1.3	41.4	1.02


All
Complete Loop Unrolling
Static Loads
Unchecked Dispatching
Static Calls
Dynamic Zero & Copy Propagation
Dynamic Dead-Assignmetn Elimination
Dynamic Strength Reduction
Internal Dynamic-to-Static Promotions
Polyvariant Division
0
1
2
3
4
5
6
7
dinero
m88ksim
mipsi
pnmconvol
viewperf: p&c
viewperf: shader
binary
chebyshev
dotproduct
query
romberg

Pros and Cons

- Pros
  - Performance improvement
  - Low overhead (staged)
- Cons
  - Annotation is time-consuming
  - Programmer has to statically predict which variables will be static
    - What if programmer is wrong?
    - How does programmer know where to look?
  - Some optimizations hurt performance
    - Mostly trial-and-error

DELI: A New Run-time Control Point

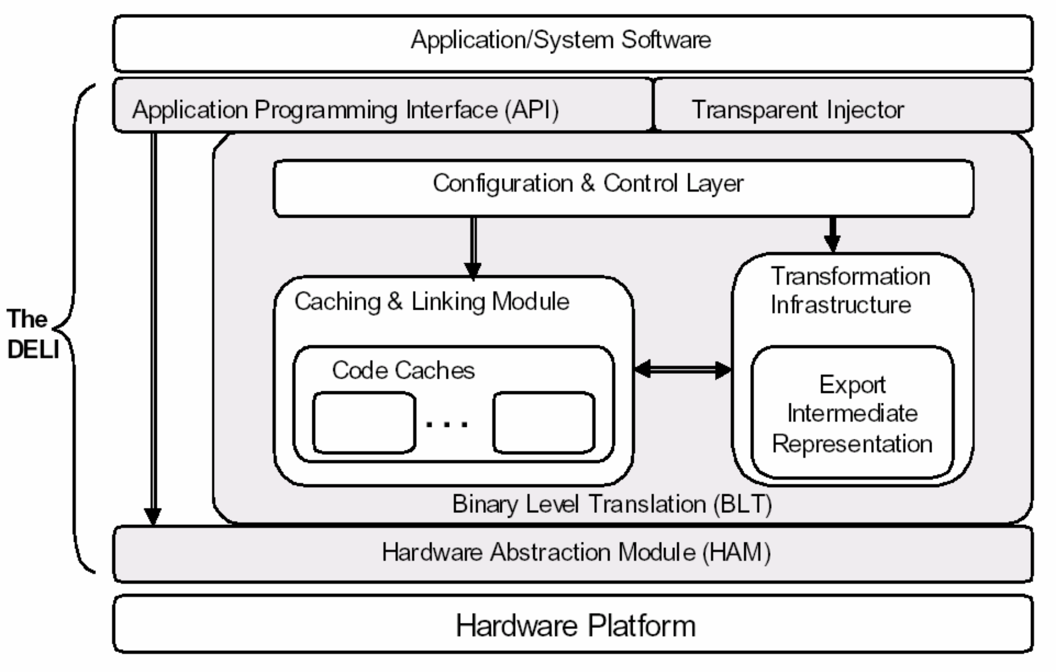
Hewlett-Packard Laboratories. MA
USA

DELI

- Successor of Dynamo system
- Extracts the underlying control functionality for caching and linking the code and exposes it to the OS and higher application layers through an explicit interface.
- Enables services such as translation, optimization, sandboxing, code patching, safety checking, hardware virtualization etc.

Overview of the DELI system

- Application Programming Interface
- Binary Level Translation
- Hardware Abstraction Modu



The DELI API

- deli_init
- deli_emit_fragment
- deli_exec_fragment
- deli_lookup_fragment
- deli_invalidate_fragment
- deli_install_callback

The DELI API

- deli_enum_fragment
- deli_setup_cache
- deli_code_cache_flush
- deli_gc
- deli_start, deli_stop

Binary Level Translation

- Code cache performance and efficiency
  - Linking fragments
  - Dynamic optimization using the DELIR
- Scheduling algorithms
  - Instruction Scheduler
  - Operation Scheduler

Hardware Abstraction Module

- Provides a virtualized view of the hardware for both the OS and DELI clients. The HAM layer components can be static
  - Static: fixed memory mappings, globally defined hooks for events such as exceptions and interrupts
  - Dynamic: manages page translation tables

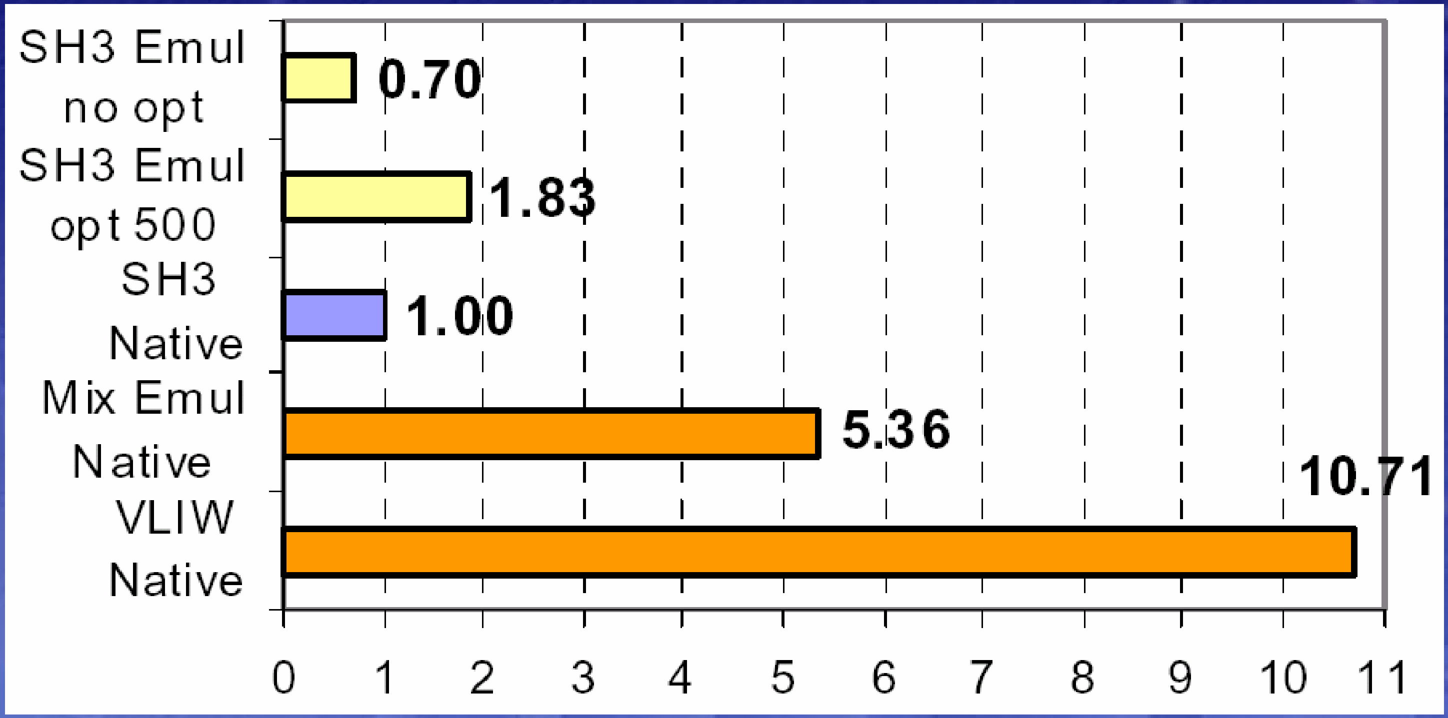
Using DELI as a Client

- The services provided by DELI can be categorized as
  - Code Manipulation
    - Dynamically patching code
    - Code decompression or decryption
  - Program Observation
    - Sandboxing
  - Emulation
    - Code Streaming

Emulating PocketPC

- Using the DELIverX prototype system which uses the Hitachi SH3 emulated interpreter, just-in-time translator for an embedded VLIW core.

Emulated vs Mixed vs Native



Critique

- Native processor is much more powerful than emulated target processor.
- No discussion on memory overhead, fragment cache size.
- Native processor does not exist and is simulated, but comparisons done vs. real machine.
- All fragments must be superblocks.

DyC Discussion

- What are the advantages of user annotations to help identify code regions or variables that benefit from dynamic optimizations?
  – It's a hard problem without help from user
  – User knows the program
- What about disadvantages?
  – Tedious, error-prone, system-specific
  – Does the user really know best?
- Can we get DyC working without user annotations?
  – Profile-directed? Information from static compiler?

More DyC Discussion

- What is the DyC equivalent for a parallel architecture?
  - Which optimizations triggered in which case?
- How do we exploit parallelism?
  - Affine expressions that are known only at run-time
  - Aliasing conditions
  - Speculative execution
  - What if memory is non-uniform?

DELI Discussion

- What other applications can you layer on top of DELI?
  - Emulation, security, code compression, …
- What are its limitations?
  - Fragments only, no profiling support
- How easy is it to port HAM to other architectures?
  - What does HAM assume about the architecture?

More DELI Discussion

- What services would you expect from a DELI-like infrastructure that targets a CMP?
  - Control over speculation, data partitioning, …
- How about a polymorphic CMP?
  - Reconfiguration, …
- How would you structure the HAM layer in a DELI for CMP?
  - What would you virtualize and how?
  - What irregular events would you handle there?

Discussion

- What do we want in a dynamic compiler infrastructure?
  - Easy to retrofit existing code?
  - How useful is explicit control over the cache?
  - How important is high-level information?
  - Programmer-specified transformations?

More Discussion

- What are the remaining dynamic compilation opportunities for CMPs and beyond?
  - Source of inspiration: what are hard problems for HW? Hard problems for static compiler?
  - Example: use dynamic compilation to hide the differences in various parallel machines
    - #nodes, latency of communication, different communication/synchronization primitives

More Discussion

- How could the static compiler help the dynamic compiler?
  - Is staged compilation always the right model?
  - What about profile information?
- What about dynamic compilation control?
  - Should control be hardwired or feedback-based?
  - How does profile information fit into the picture?