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

Annotated Program Source

Traditional Optimizations

Binding-Time Analysis (BTA)

Dynamic-Compiler Generator

Program Input

Statically Generated Code

Dynamic Compiler

Dynamically Generated Code

Static Compile Time

Run Time

May 15, 2003
Dynamic Compilation II: DyC and DELI
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

```c
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;
        }
    }
}
```
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; }}}
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;
        ...
        outbuf [irow][icol] = sum; }}
}
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

<table>
<thead>
<tr>
<th>0</th>
<th>-1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Sharpen**
- **Emboss**
- **Example**

May 15, 2003
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

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

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

<table>
<thead>
<tr>
<th>Program</th>
<th>Asymptotic Speedup</th>
<th>Execution Time in the Dynamic Regions (% of total static execution)</th>
<th>Average Whole Program Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>dinero</td>
<td>1.7</td>
<td>49.9</td>
<td>1.5</td>
</tr>
<tr>
<td>m88ksim</td>
<td>3.7</td>
<td>9.8</td>
<td>1.05</td>
</tr>
<tr>
<td>mipsi</td>
<td>5.0</td>
<td>~100</td>
<td>4.6</td>
</tr>
<tr>
<td>pnmcconvol</td>
<td>3.1</td>
<td>83.8</td>
<td>3.0</td>
</tr>
<tr>
<td>viewperf</td>
<td>1.3</td>
<td>41.4</td>
<td>1.02</td>
</tr>
</tbody>
</table>
May 15, 2003

Dynamic Compilation II:
DyC and DELI

18
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 Module
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

- SH3 Emul no opt: 0.70
- SH3 Emul opt 500: 1.83
- Native: 1.00
- Mix Emul: 5.36
- VLIW: 10.71
- 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?