Wiggins/Redstone:
An On-line Program Specializer

Dean Deaver
Rick Gorton
Norm Rubin
{dean.deaver,rick.gorton,norm.rubin}@compaq.com
W/R is a Software System That:

- Makes arbitrary binary applications run faster without requiring any work from the programmer
- Aggressively optimizes/specialize an application for a particular use on a particular machine
- Moves optimization/compilation closer to the actual use of the program
The W/R System Architecture

1. The agent - A modified loader/launcher that starts the system
2. A low overhead, hardware based sampler
3. A trace builder that finds and instruments parts of a program
4. Optimizer/specializer (works on superblocks)
While the program is running {
    1. Identify a hot instruction
    2. Build a trace containing the instruction
    3. Instrument the trace
    4. Specialize the trace
    5. Optimize the trace
}

Step 1 is hardware, 2-5 are software
Agent

- A special loader
- Adds code to an image when started. This code contains the profiler and optimizer
- The agent is shared over applications
- The agent knows about the actual platform, so old programs can run on new platforms
- Allows us to add new optimizations to old programs
We use a hardware PC sampler to find “hot” seed instructions:

- The sampler is a source of frequent interrupts
- Look for frequent values of program counter at interrupt time
- Code is based on DCPI

Approach works on out-of-order machines such as 21264
Given a seed instruction
- Copy it and the remainder of the block to a side buffer
- Add instrumentation code, guards to insure correctness, branch back
- Patch the image to branch to the copy
- After the instrumentation code finds the most common successor extend the copy

Copied instructions form a superblock
Effectively a lazy instruction trace constructor
Specializes "hot" traces using machine-specific information. Introduce guards as necessary
Exploits temporal info
Analyzes what to monitor
Performs architectural and micro-architectural optimizations (byte/word loads and stores on alpha)
Applications will continuously monitor themselves and perform self-improvements whenever necessary
Advantages

- The application carries no machine-specific information
- Can update the agent to incorporate new optimization techniques as they become available
- Programs compiled using generic or specific features run faster on specific programs run faster on 21264, ...
Povray - a freely available rendering package

Image “matches.Pov”
- 2 billion calls to power(x,y)
- If you perform three levels of inline on the frequent path you find that y = 8.0
- Calls to power() are on the frequent path 95% of the time
How Many Traces?

- Typically less than 16 traces at a time
- Traces contain several hundred instructions
- Traces often account for 50-90% of the run time of an image
- Traces are removed as the computation evolves
PovRay shapes.pov Demo
Percent of Time on Traces

Time

Percent of Time on Traces

Compaq
Better answers
Characteristics of Traces (shapes.pov)

- Povray: 661069 static instructions
- Traces
  - 7 traces total
  - 1586 instructions (0.239%)
  - 819 unique instructions (0.123%)
Characteristics of Traces

- Inter-procedural
  - Often 2-4 levels deep
- Can include one loop
  - But may include many unrolled loops
- May be up to 2000 instructions long
  - Often 300-500 instructions
  - Long enough to insert pre-fetch instructions
- Need not stop at a register transfer, return, or call site
Conditional Branches (Cbrs)

- 76 unique cbrs
- 115 instances of a CBR show up on various traces
- Trace probabilities vs. Aggregate probabilities
  - Correlated branches
  - Temporal effects
- 5-10% of branches have multiple instances with reversed directions
### Cbrs Vs Static Branch Probs

<table>
<thead>
<tr>
<th>Branch</th>
<th>Trace</th>
<th>Probability</th>
<th>Aggregate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12003cc48</td>
<td>3</td>
<td>1.00</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>0x12003cc74</td>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0x12003cc9c</td>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.00</td>
<td></td>
</tr>
</tbody>
</table>
Temporal Effects

- A single program using one data set can show phases, which may not be apparent in the source code.
- Different phases require different optimizations.
- E.G.. Compress (SPEC95) -
  - For each data item - look it up in hash table
  - Initially most items are not in table
  - Later most items are in table
Sunsethf.pov

Trace construction (Time)
Temporal Effects
What Don’t We Do?

- W/R works on applications not system kernels
- Does not modify OS components
- Does not modify program memory layout
- Does not work with device drivers
What about Linux?

- Need a driver to sample performance counters
- We only do Alpha
- Are Linux users more forgiving?
- Open source opportunity?
- What about clusters?
Final Comments

- Wiggins/Redstone is the software analog of a trace processor
- Runs on stock hardware/stock OS
- Optimizes/specializes binary images
- Runs on-line
- Captures temporal effects
- One tool, in a more adaptive computing model?
- Return to self-modifying code?