Hybrid (Static and Dynamic) Compiler Optimization

Silvius Rus, Carson Brownlee, and Lawrence Rauchwerger
cbrownle@gonzaga.edu, {silviusr,rwerger}@cs.tamu.edu
Parasol Lab, Department of Computer Science, Texas A&M University, http://parasol.tamu.edu
Hybrid Analysis (Static and Dynamic) of Memory Reference Patterns

Motivation

• Industry is committed to multithreaded architectures (IBM Cell, Intel Pentium D, Sun Niagara), but application software generally cannot utilize them.

• System resource or configuration changes dramatically, e.g. SoCs (system-on-a-chip) and ASICs (Application-Specific Integrated Circuit), so writing chip specific code is very hard for users.
Hybrid Analysis (Static and Dynamic) of Memory Reference Patterns

• Application software lags behind
  • Lack of explicitly multithreaded applications
  • Difficult to write, port, and test by hand

• Compiler-based Automatic Parallelization
  • Dynamic behavior requires dynamic analysis

• Compile-time Analysis:
  • Overly conservative, poor performance improvement

• Run-time Analysis:
  • Although exact, its high cost may offset the benefits
  • Wasteful: ignores partial compile-time results
Hybrid Analysis (Static and Dynamic) of Memory Reference Patterns

Hybrid Analysis Overview

**IDEA:** At run-time, perform the part of the analysis that could not be completed at compile-time

- Perform aggressive compile-time analysis
- Rather than taking conservative decisions, postpone the analysis parts that depend on dynamic information (input values)
- Formulate efficient run-time checks based on partial results

<table>
<thead>
<tr>
<th>Compile-time Analysis</th>
<th>Hybrid Analysis</th>
<th>Run-time Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STATIC</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symbolic analysis</td>
<td>Symbolic analysis</td>
<td></td>
</tr>
<tr>
<td><strong>DYNAMIC</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Continue analysis with actual values</td>
<td>Full reference-by-reference analysis</td>
<td></td>
</tr>
</tbody>
</table>
Hybrid Analysis (Static and Dynamic) of Memory Reference Patterns

Example

```fortran
READ *, N
DO j=1,N
   a(j)=a(j+40)
ENDDO
```

Figure 1. Sample Code

Figure 2. The Dependence Set as intersection of READ and WRITE descriptors

Figure 3. After compile-time analysis

Figure 4. Reference by reference test

Figure 5. Hybrid test
Hybrid Analysis (Static and Dynamic) of Memory Reference Patterns

- **RT_LMAD** = Run Time Linear Memory Access Descriptor
  - Closed form descriptor for memory location sets
  - Persistent between compile time and run time

- Analysis and optimization using RT_LMADs
  - Data Independence: $D_{\text{read}} \cap D_{\text{write}} = \phi$
  - Removal of anti dependences: $D_{\text{read}} \subseteq D_{\text{write}}$
  - Checkpoint memory exclusion: $D_{\text{write}}^{1} \subseteq D_{\text{write}}^{2}$

Anatomy of an RT_LMAD
Hybrid Analysis (Static and Dynamic) of Memory Reference Patterns

Static and Dynamic Analysis

• Analysis of programs
  • Aggregate information as RT_LMADs
  • Take static decisions whenever possible

• Synthesis of dynamic tests
  • Embed RT_LMADs as executable code

• Run time
  • Complete analysis phase: evaluate RT_LMADs
  • Take optimization decisions
Hybrid Analysis (Static and Dynamic) of Memory Reference Patterns

Performance Improvement

Whole-application speedup after automatic parallelization
Graph visualization is an excellent way to understand compilation. However, the graphs can be hard to view given their size and shape are mostly static. DIVE was created to allow both the viewing of these graphs interactively (zooming, panning) and as a way to manipulate the graphs at run-time.

**Client**

(Flash, ActionScript, HTML, JavaScript)

**Graph Server**

(C, C++, STAPL, Graphviz)

**Client Functions:**
- Load Graph Image from server
- Load Layout from server
- Display the image

**Program**

```plaintext
PROGRAM main
  READ *, N
  DO j = 1, N
    a(j) = a(j+40)
  ENDDO
```

**Graph Server Functions:**
- CreateGraphImage: The server will take in a graph in GXL format and return an image (jpg/swf/etc).
- ExpandNode(n): Server will add the children of n into the display graph.
- Other similar graph manipulations.

**Server:** Stores, draws and lays out graphs. It uses GraphViz to display graphs stored in the Graph Exchange Language GXL (an XML dialect).

**Commands:**
- expand, add

**Graphs:**
- Image URL, layout
DO i = 1, nnped
IF (icond(i).LE.0)
   DO j = 1, 5
      XE(j,i) = 0
   ENDDO
ENDDO
ENDDO

Code design and review, program composition, compiler analysis understanding and monitoring
Dynamic optimization visualization, performance monitoring, symbolic high-level debugging