Compiler Optimizations for Speculative Multi-Threading Processors
Jenq Kuen Lee
Dept. of Computer Science
National Tsing Hua University
Hsinchu, Taiwan

(Joint work with Peng-Sheng Chen, Ming-Yu Hung, Yuan-Shin Hwang, and Roy Dz-Ching Ju)
Outline

- Motivation
- Multi-Threading Processor Overview
- Probabilistic Points-to Analysis
- Data Dependence Probability
- Experimental Results
- Conclusion
Compiler Issues for SIP Mall

1. Meta level attribute and description

   Especially for processors

2. DSE by Compilers

3. Software integration
Multithreading Processor

• Academic community
  – Stanford: Hydra, MIT: Raw, Wisconsin: Multiscalar, Minnesota: Superthreading, …

• General-purpose processor
  – Intel Xeon Hyper-threading

• Network processor
  – High speed: CMP (Single chip Multiple Processors)
  – Packet Level Parallelism: SMT (Simultaneous Multiple Threads)
  – ex: Intel IXP 1200, 6 multi-threaded cores.
Case Study: Intel IXP1200
Multithreading DSP Processors

- Multithreaded DSP
  - Expect perform multi-functions (tasks) at the same time.
  - ex: MMS (Multimedia Messaging Service)
    • Multimedia (MP3, MPEG4) + Comm. (GSM, Bluetooth)
  - ex: Metagence META DSP

Conventional approach

Meta approach
Multithreading Processor (cont’d)

– Sandbridge’s SB9600
  • Targeted at 3G wireless, multimedia devices.
  • Execute 8 threads simultaneously.

– MemoryLogix MLX1
  • A Small, Multithreaded 586-Compatible Core
  • Targeted at smart mobile devices.
  • Execute 3 threads simultaneously.
Motivation

- Speculative multithreading (**SpMT**)
  - Solve dependence relationship at runtime
  - Exploit available parallelism of the program
  - mis-speculation: high recovery penalty

![Diagram showing speculative multithreading with threads being discarded and code recovered](image)
Motivation (cont’d)

- Should compiler use speculation or not?
- Cost Model

\[ L_o > L_s + \sum_{set} (V_{freq}(set) \times (O_r^{set} + L_c^{set})) \times P_{dep}(set) \times E \]

- \textit{set}: each possible violation relationship
- Assume the violation relationships are independent
Motivation (cont’d)

Should compiler use speculation or not?

Construct cost model

Need data dependence probability

Need points-to relationship probability
Motivation (cont’d)

- Guide compiler use speculation or not?
- Evaluate cost model
- Compute data dependence probability
- Compute points-to relationship probability
Probabilistic Points-to Analysis

Definition

- At every program point $s$, for every points-to relationship, say $(p, v)$, a probabilistic points-to relationship $(p, v, P)$ is computed:

$$P = \frac{E(s, \langle p, v \rangle)}{E(s)}$$

$E(s)$: the number of times $s$ is visited.

$E(s,(p,v))$: the number of times $(p,v)$ holds at $s$. 
Analysis Algorithm

- Iterative data flow framework
- Transfer function
  - computes points-to sets for each program statement
    \[ \text{OUT}_S = F_S(\text{IN}_S) \]
  - Probability function
    - computes the probability of each element
    \[
    P\left(s, \langle p, v \rangle \right) \overset{\text{def}}{=} \frac{E\left(s, \langle p, v \rangle \right)}{E(s)}
    \]
### Basic Pointer Statements

- Basic pointer assignment statements

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p=\text{nil}</td>
<td></td>
</tr>
<tr>
<td>*x=\text{nil}</td>
<td></td>
</tr>
<tr>
<td>p=&amp;q</td>
<td></td>
</tr>
<tr>
<td>p=q</td>
<td></td>
</tr>
<tr>
<td>p=*q</td>
<td></td>
</tr>
<tr>
<td>*x=q</td>
<td></td>
</tr>
</tbody>
</table>
Probability Function \( P(S_{out}, \langle p, v \rangle) \)

- **S: \( p=\text{nil} \)**
  
  \[ P_x = 0 \] \( \rightarrow \) \( v \)  
  \[ \text{nil} \]

- **S: \( *x=\text{nil} \)**
  
  \[ P_1 \] \( \rightarrow \) \( p \)  
  \[ p \]

  \[ P_x = (1 - P_1) \cdot P_x \] \( \rightarrow \) \( v \)  
  \[ \text{nil} \]
**Probability Function**  \( P(S_{\text{out}}, \langle p, v \rangle) \)

\[ S: p=\& q \]

\[ q \equiv v \]

\[ p \quad P_x = 1 \rightarrow v \]

\[ q \not\equiv v \]

\[ p \quad P_x = 0 \rightarrow v \]

\[ S: p=q \]

\[ P_x = P_1 \rightarrow v \]

\[ q \]

\[ P_1 \]
Probability Function \( P(S_{out}, \langle p, v \rangle) \)

- \( S: p=*q \)
- \( S: *x=q \)

\[ \sum_i P_{xi} \times P_{yi} \]
Probability Function (cont’d)

- Meet operation

\[ P\left(B_{1_{out}}, \langle p, v \rangle \right) \times E(B_1) + P\left(B_{2_{out}}, \langle p, v \rangle \right) \times E(B_2) \]

\[ \frac{E(B_1) + E(B_2)}{E(B_1) + E(B_2)} \]
Probability Function (cont’d)

- **IF-then-ELSE construct**
  \[ P\left( \text{Join}_{in}, \langle p, v \rangle \right) = P\left( \text{Then}_{out}, \langle p, v \rangle \right) \cap P\left( \text{Else}_{out}, \langle p, v \rangle \right) \]

- **Loop construct**
  \[
P\left( \text{Header}_{in}, \text{IN}_{\text{Header}} \right) \\
  = P\left( \text{B0}_{out}, \text{OUT}_{B0} \right) \cap P\left( \text{B}[1]_{out}, \text{OUT}_{B[1]} \right) \cap L \\
  = P\left( \text{B0}_{out}, \text{OUT}_{B0} \right) \cap \left( \cap_{i=1}^{\infty} P\left( \text{B}[i]_{out}, \text{OUT}_{B[i]} \right) \right) \\
  = P\left( \text{B0}_{out}, \text{OUT}_{B0} \right) \cap P\left( \text{B}_{out}, \text{OUT}_{B} \right)
  \]
Example 1

\[ P(Header_{in}, IN_{Header}) = P(B_{0out}, OUT_{B0}) \times P(B_{out}, OUT_{B}) \]

\[ P = \frac{1 \times 1 + 9 \times 0.8P}{1 + 9} \]

\[ P = 0.357 \]
Data Dependence Probability

Definition

\[ P_{S_1S_2} = P_{S_1} \times P_{S_2} \times P_{S,S} \left( s, \left\langle p_{S_1}, q_{S_2} \right\rangle \right) \]

\[ P_{S_1S_2} = E \left( s_2, \left( p_{S_1} \rightarrow q_{S_2} \right) \right) / E \left( S_2 \right) \]
Experimental Results

- Verify the **precision** and **accuracy** of the probabilistic points-to analysis (PPA).
- Evaluate the **impact of performance** with incorporation of dependence analysis and PPA on **SpMT**.
Experimental Results (cont’d)

- The structure of PPA compiler

![Diagram of PPA compiler structure]

- C program
- SUIF1 System
- Low SUIF Intermediate Format
- SPAN (location set front-end)
- Probabilistic Points-to Analyzer (PPA)
- MachSUIF Control-Flow Graph Library
- GNU Scientific Library
- Boost Graph Library
- GiNaC Symbolic Algebra System
- Points-to relationship probability for each basic block
- CFG represented by VCG
- PPA-S: static probability assigned to each outgoing edge of CFG
- PPA-P: runtime path profiling information
Superthreading Architecture

Thread i

<table>
<thead>
<tr>
<th>Stage</th>
<th>Thread i</th>
<th>Thread i+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuation Stage</td>
<td>Thread fork &amp; forward continuation variables</td>
<td>Thread fork &amp; forward continuation variables</td>
</tr>
<tr>
<td></td>
<td>target store address</td>
<td>target store address</td>
</tr>
<tr>
<td></td>
<td>TSAG_DONE flag</td>
<td>TSAG_DONE flag</td>
</tr>
<tr>
<td>Target Store Address</td>
<td>target store address &amp; data</td>
<td>target store address &amp; data</td>
</tr>
<tr>
<td>Generation Stage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computation Stage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write Back Stage</td>
<td>WB_DONE flag</td>
<td>WB_DONE flag</td>
</tr>
</tbody>
</table>

Thread fork & forward continuation variables
TSAG_DONE flag
Target store address & data
WB_DONE flag
Experimental Results (cont’d)
Experimental Results (cont’d)
Experimental Results (cont’d)

- Applications on SpMT
  - Simulator: SIMCA (ARCTiC Lab., Minnestoa)
  - Superthreading architecture
  - Pointer-induced loop-carry data dependence

- To evaluate compiler-directed speculation
  - Sequential Execution
  - Speculation
  - Probabilistic Speculation
Experimental Results (cont’d)

Benchmark Programs

- data retrieval
- em3d
- malloc
- map
- shuffle
- 990127-1

SpeedUp

- sequential
- speculation
- Probabilistic Speculation
Conclusion

- PPA application
  - Guide compiler to use speculation mechanism
    - SpMT
    - Data Speculation
  - Details in our paper (ACM PPoPP 2003).

- Future Work
  - Incorporate array and pointer variables
  - More experimental results on large programs