ACR: Enabling Computation Reuse through Approximate Computing

Xin He, Guihai Yan, Yinhe Han and Xiaowei Li

State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences (ICT, CAS)
Power efficiency

➢ High energy efficiency is required

Video gaming

Robotics

Image processing

Machine learning
Approximate computing

- Exploiting **tradeoffs** between quality and complexity

**DCT Image Processing**

- Original method
- Approximate method

Quality: **2.82%** Reduction
Performance: **31.44%** Improvement

**Machine Learning**

Quality: **1.3%** Reduction
Performance: **4.97X** Improvement
Conventional Computation Reuse scheme

Brief workflow of traditional computation reuse scheme

Reuse requirement: If and only if input values of incoming calls equal to the ones in history.
Efficacy of Conventional Reuse Scheme

- Proportion of reuse opportunities exploited

Reuse potential:
As much as 99%

Conventional reuse:
As low as 27%

In approximate application, there’s great reuse potential

Tight reuse requirement prevents CONVENTIONAL CRU from achieving high speedup
To enable computation reuse for approximate computing, we need:

**Similarity Quantification**
- A criterion to measure the similarity between different

**Branches Resolving**
- Reduce the latency of packets according to traffic heterogeneity
Workflow of Proposed ACR

- Two extra steps should be included into approximate computing process.

1. Deciding the computation pattern
2. Calculating the similarity

Diagram:
- Branch Resolving
- Similarity quantification
- Computation buffer
- Incoming function call with inputs $i_1, i_2, \ldots, i_n$
- Example: $\text{BlackScholes}()$ have six input parameters.

Decision:
- $\text{Deciding the computation pattern}$
- $\text{Calculating the similarity}$
Prior approximate reuse scheme (i.e. RACB scheme from islped 2005)

Main idea: masking LSBs of values of inputs, and requiring the MSBs to be equal for approximate reuse

Pros:
1. Easy to implement
2. Effective for specific applications

Cons:
1. Too arbitrary
2. Limited speedup
Capture the relation between inputs and outputs and estimate the significance of inputs.

- Code region or function at runtime
- Perturbation analysis cannot be applied

Use executed computation to model the code region or function

- We have:
  1) Values of inputs
  2) Values of outputs
  3) Branch decisions

from compiler explicit markers
Proposed significance aware approximate reuse

- Linear regression based statistical technique to obtain significance of inputs

E.g. inversek2j() benchmark:

\[ z = \cos((x^2 + y^2 - 0.5)/0.5) \]  
\[ \text{Output} = \sin((y \times (0.5 + 0.5 \times \cos(z)) - x \times 0.5 \times \sin(z))/(x^2 + y^2)) \]
Resolving conditional branches

- Logistic regression based statistical technique to modeling conditional branches in function
- Then the model is used to obtain branch decision of incoming
Speculative branch prediction

Actual conditional branch:

$$if (q_0 - p_0 < 20)$$

Predicted conditional branch:

$$-0.96p_0 + 0.08p_1 - 0.06p_2 + q_0 - 0.07q_1 + 0.03q_2 = 20.09$$
Overview of proposed ACR scheme

1. Trained with computations executed in history

For an incoming Computation, Its computation pattern is specified first

Then, Its similarity with history computation is obtained
Benchmarks

- Axbench: a benchmark designed for approximate computing from Georgia tech

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
<th>N.in</th>
<th>N.br</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blackscholes</td>
<td>Pricing a portfolio of options with the Black-Scholes equation</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Inversek2j</td>
<td>Robotic: Inverse kinematics for 2-joint arm</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Sobel</td>
<td>Sobel edge detector in Image Processing</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>Jmeint</td>
<td>Triangle intersection detection in 3D gaming</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>H264</td>
<td>Loop filter in h264</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>
The percentage of computation effort
Storage

- Area overhead:

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Blackscholes</th>
<th>Inversek2j</th>
<th>Sobel</th>
<th>Jmeint</th>
<th>H264</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage Cost(kB)</td>
<td>65.73</td>
<td>40.08</td>
<td>29.05</td>
<td>37.14</td>
<td>42.38</td>
</tr>
</tbody>
</table>
Conclusion

- Approximate computing can exploit the potential of computation reuse
- Not all error-tolerant application are suitable for approximate computation reuse
- Error-tolerant applications which benefit most from ACR would be:
  - Time consuming: contains complex function like sin(), cos(), exponential function
  - Contain small number of input parameters
  - Have a few number of conditional branches
Thanks for listening!

Question?