Overcoming the Data-flow limit with Structural Approximation

Vignesh Balaji          Brandon Lucia          Radu Marculescu
Carnegie Mellon University
Outline

❖ Trends in Approximate Computing
❖ Structural Approximation
❖ Making Structural Approximation viable
Correctness is *costly*

AND

Precision is not always *required*
Successful Domains for Approximation

Most existing approximations typically target only numeric values

Another Interesting Domain - Parallel Computing

- Parallel programs communicate to enforce *correctness*:
  - Synchronization primitives (locks, barriers, mutexes, etc.)
  - Cache Coherence
- Communication limits *scalability*

Should we still communicate for values that are not *critical* for the app’s correctness?
Removing Communication via Approximation

Prior work has successfully removed synchronization for such \textit{values}

\textit{Rely, Relaxing Synchronization for Performance and Insight, Dancing With Uncertainty, Unsynchronized Techniques for Approximate Parallel Computing, ...}

Are such \textit{value} approximations enough? Or is there a need for something more?
Quantifying the Performance Limit

Simulated the maximum performance possible with Value Approximation:

- Removed *all* locks guarding values
- Removed all *effects* of cache coherence
- Removed the *cost* of performing reduction operations

Definition of correctness for the limit study -

The application should not *crash*
Lock(table[x].mutex);

Value Approximation

Unlock(table[x].mutex);

Approximating Key-Value Store

Key-Value Store

- Amdahl's limit
- Value Approximation
- Normal
Approximating K-Means

Step 1: Assign points to the closest cluster

Step 2: Compute means of the new clusters

K-Means

<table>
<thead>
<tr>
<th>Speedup</th>
<th>4 threads</th>
<th>16 threads</th>
<th>64 threads</th>
<th>256 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amdahl's limit</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Value/Convergence Approx.</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Normal</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Value Approximation Works Well

- Approximating data values took performance close to Amdahl’s limit

- Previous two applications only shared numerical values

What if we have applications that share more than just numerical values?
Breadth First Search

Data structure that holds the children nodes of a frontier is shared.
Single Source Shortest Path

Data structure maintaining nodes closest to the source is shared
End of the Road for Value Approximation

What’s different in these applications?

These application *share data structures* in addition to values

What if we remove synchronization for operations guarding data structures?

Removing synchronization for data structures leaves them *inconsistent*

How do we reason about output quality now?
Outline

❖ Trends in Approximate Computing
❖ Structural Approximation
❖ Making Structural Approximation viable
## Overview of Structural Approximation

<table>
<thead>
<tr>
<th>Precise Output</th>
<th>Output after Value Approximation</th>
<th>Output after Structural Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="precise_output" alt="Diagram" /></td>
<td><img src="value_approximation" alt="Diagram" /></td>
<td><img src="structural_approximation" alt="Diagram" /></td>
</tr>
</tbody>
</table>

1. **Precise Output**: The output before any approximation is applied.
2. **Output after Value Approximation**: The result after approximating the values but keeping the structure intact.
3. **Output after Structural Approximation**: The final output after approximating both values and structure, resulting in `NULL` for the root node in all cases.
Mechanism of Structural Approximation

Precise Version

Tid 1

Tid 2
Mechanism of Structural Approximation

Precise Version

Tid 1  Tid 2
Mechanism of Structural Approximation
Mechanism of Structural Approximation

Precise Version

Approximate Version

Tid 1

private cache

Tid 1

Tid 2

Tid 2
Mechanism of Structural Approximation
Mechanism of Structural Approximation

How can we ensure *usable* results from Structural Approximation?
Outline

❖ Trends in Approximate Computing
❖ Structural Approximation
❖ Making Structural Approximation viable
Existing Definition of Correctness

- Value based approximations use *numerical distance* to measure error.

\[
\text{Metric} = (\text{Output}_{\text{precise}} - \text{Output}_{\text{approximate}})^2
\]

- Makes sense for value approximation.

We cannot use this error definition for Structural Approximation!
New Definition of Correctness

The new definition should:

➢ Accommodate *temporary* inconsistency in data structures
➢ *Tolerate* the loss of a few elements in data structures

Need mechanisms to *enforce* this correctness definition
Resilient Operators

- Operators that *successfully* service queries even on inconsistent data structures
- Similar to the concept of Defensive Programming

```c
int risky_visitor(int size){
    while(i < size){
        do_work(node->value);  //Fails if elements dropped from list
        node = node->next;
        i++;
    }
}
```

```c
int secure_visitor(int size){
    while(i < size){
        if (node != NULL){
            do_work(node->value);  //saved from //NULL ptr exception
            node = node->next;
            i++;
        } else
            break;
    }
}```

Handle all failures that are possible due to Structural Approximation
Data Structure Repair

- Major challenge - How to implement *low-cost* repairs?
  - Offload repair to cloud?
Role of SA in the Future of Approximate Computing
Thank You