Can We Make It Faster? Efficient May-Happen-in-Parallel Analysis Revisited

Congming Chen, Wei Huo, Lung Li
Xiaobing Feng, Kai Xing

State Key Laboratory of Computer Architecture
Institute of Computing Technology, Chinese Academy of Sciences.
Outline

- Background
- Problems Formulation
- Solutions: design and implementation
- Evaluation
- Conclusion
What is May-Happen-in-Parallel (MHP) analysis?

void *func2(void* arg){
    ....... // part D
}

void *func1(void* arg){
    ....... // part C
}

int main(){
    pthread_t th1, th2;
    pthread_create(&th1, NULL, func1, NULL);
    ....... // part A
    pthread_join(th1, NULL);
    pthread_create(&th2, NULL, func2, NULL);
    ....... // part B
    pthread_join(th2, NULL);
    return 0;
}
What is May-Happen-in-Parallel (MHP) analysis?

More complex Inter-threaded Control Flow Graph
Background

• Classical May-Happen-in-Parallel (MHP) analysis method
  – Base on series of predefined iterative data flow equations (IDFB approaches)
  – Data flow equations are designed according to the concurrent syntax of certain keywords (eg. In Java, keywords are including: start, join, notify, notifyall, wait)
  – Program representation is the so-called Parallel Execution Graph (PEG), which is constructed from the inter-threaded control flow graph
  – Each node $n$ in PEG has two sets:
    • A M$(n)$ set: all nodes which can happen in parallel with $n$
    • A OUT$(n)$ set: all nodes which can happen in parallel with the successors of $n$
Background

Algorithm overview

Basic Data Equation

\[ OUT(n) = (M(n) \cup GEN(n)) \setminus KILL(n) \]

**Algorithm Overview**

**Basic Data Equation**

\[ OUT(n) = (M(n) \cup GEN(n)) \setminus KILL(n) \]

**Input:** PEG=<N, E>

**Output:** each node n in PEG get its M(n) set

**Work-list** W  all start nodes in PEG

compute KILL sets for all join nodes;

compute GEN sets for all start nodes.

while(W is not empty) {  

| n  get from W  

| \[ M_{old}(n)  M(n) \]

| \[ OUT_{old}(n)  OUT(n) \]

compute M(n) according to equation

\[ M_{diff} = M_{old}(n) \cap M(n) \]

for each element e in M_{diff}

| \[ n \rightarrow M(e) \]

| add e into W \hspace{1cm} // Add node to W

end for

compute OUT(n) according to equation

if(OUT(n) \neq OUT_{old}(n))

| add successors of n into W \hspace{1cm} // Add node to W

end if
Outline

- Background
- Problems Formulation
- Solutions: design and implementation
- Evaluation
- Conclusion
In the state-of-art IDFB approach: getting node from work-list and adding nodes to work-list are in a First-In-First-Out (FIFO) order.

Problem: A node may be processed many times even though its MHP information has not been updated! Which will lead to a lot of redundant computation.
Problems Formulation

Key Observations

- **Non-topological Order:**
  A node is processed after its successor nodes. As a result, the successor nodes need to be processed again if the OUT set of current node is updated.

- **Eager Update:**
  According to the symmetry requirement of MHP analysis in the original algorithm, each time a node $n$ is added into the $M(m)$, $m$ will be added into $M(n)$ as well, and $n$ will be added into the work-list.
Outline

- Background
- Problems Formulation
- Solutions: design and implementation
- Evaluation
- Conclusion
Solutions: design and implementation

**Definition: Parallel Level (PL)**

∀ \( t \in \text{PEG} \), \( t \) is a thread. We assign a number to \( t \) as the parallel level of \( t \), which is represented as \( \text{PL}(t) \). PL is defined as follows:

1) Given two threads \( t_1, t_2 \in \text{PEG} \), if thread \( t_2 \) is created by thread \( t_1 \), then \( \text{PL}(t_2) = \text{PL}(t_1) + 1 \).

2) Suppose \( \text{main} \) stands for the main thread of the program, then we define \( \text{PL}(\text{main})=0 \). Hence, considering 1), we have \( \forall t \in \text{PEG}, \text{PL}(t) \geq \text{PL}(\text{main}) \).

**Definition: Strict Topological Order (STOPO)**

Algorithm details can be found in the paper.

*Start from the root node of PEG, get intra-threaded topological order for threads with different PL.

*Preserve the sorted node in a list, which will be used in the iteration process.

*STOPO may not be unique, however, same results will be produced based on different STOPOs.
Solutions: design and implementation

Strict topological order of the motivational example:

Our implementation:

1) Get node from work-list and add node to work-list according to STOPO – To avoid non-topological order

2) Keep traversing the sorted list over until the work-list is empty – To avoid eager update
Problems Formulation

- Again: the motivational example

STOP1: Process each node just according to the strict topological order

STOP2: Our final solution
Outline

- Background
- Problems Formulation
- Solutions: design and implementation
- Evaluation
- Conclusion
Four approaches have been implemented in our experiment, including:

1) FIFO: The original approach
2) STOP1: Introduced for the sake of comparison
3) STOP2: Our solution
4) TCT: State-of-art non-IDFB approach

**Fig**: Relative speedup between different approaches

- FIFO/STOP2: 29.02x
- TCT/STOP2: 10.00x
# Evaluation

<table>
<thead>
<tr>
<th>TEST CASES</th>
<th>KLOC</th>
<th>T#</th>
<th>N#</th>
<th>E#</th>
<th>IDFB MHP TIME(s)</th>
<th>NODE VISITING NUM MAX#/AVERAGE#</th>
<th>TCT TIME(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FIFO STOP1 STOP2</td>
<td>FIFO STOP1 STOP2</td>
<td></td>
</tr>
<tr>
<td>fft</td>
<td>1.47</td>
<td>4</td>
<td>1602</td>
<td>1946</td>
<td>11.71 22.89 0.75</td>
<td>229/146 1069/481 2/1</td>
<td>4.35</td>
</tr>
<tr>
<td>radix</td>
<td>1.43</td>
<td>6</td>
<td>1885</td>
<td>2295</td>
<td>10.8   42.34 1.17</td>
<td>178/95 1431/660 2/1</td>
<td>5.07</td>
</tr>
<tr>
<td>lu-con</td>
<td>1.39</td>
<td>5</td>
<td>1250</td>
<td>1529</td>
<td>3.64   10.88 0.27</td>
<td>129/70 817/340 2/1</td>
<td>2.29</td>
</tr>
<tr>
<td>lu-non</td>
<td>1.16</td>
<td>5</td>
<td>1186</td>
<td>1451</td>
<td>3.61   10.71 0.27</td>
<td>129/74 817/359 2/1</td>
<td>1.98</td>
</tr>
<tr>
<td>cholesky</td>
<td>5.69</td>
<td>2</td>
<td>23740</td>
<td>28711</td>
<td>5062   17431.98 54.36</td>
<td>466/310 9819/4063 2/1</td>
<td>692.32</td>
</tr>
<tr>
<td>barnes</td>
<td>3.51</td>
<td>4</td>
<td>1396</td>
<td>1687</td>
<td>7.29   18.11 0.42</td>
<td>208/120 1081/501 2/1</td>
<td>2.67</td>
</tr>
<tr>
<td>fmm</td>
<td>5.43</td>
<td>2</td>
<td>9102</td>
<td>10526</td>
<td>534.7  1471.67 10.9</td>
<td>428/238 4529/2256 2/1</td>
<td>116.32</td>
</tr>
<tr>
<td>ocean-con</td>
<td>8.18</td>
<td>2</td>
<td>5255</td>
<td>6437</td>
<td>312.7  319.2 3.53</td>
<td>947/551 2529/1219 2/1</td>
<td>35.29</td>
</tr>
<tr>
<td>ocean-non</td>
<td>6.25</td>
<td>2</td>
<td>5066</td>
<td>6222</td>
<td>274.46 279.42 3.23</td>
<td>943/532 2400/1139 2/1</td>
<td>33</td>
</tr>
<tr>
<td>raytrace</td>
<td>11.04</td>
<td>2</td>
<td>5714</td>
<td>7081</td>
<td>10.51  32.83 0.5</td>
<td>113/24 769/104 2/1</td>
<td>51.45</td>
</tr>
<tr>
<td>volrend</td>
<td>5.49</td>
<td>2</td>
<td>1130</td>
<td>1347</td>
<td>2.81   3.41 0.12</td>
<td>205/80 408/152 2/1</td>
<td>0.93</td>
</tr>
<tr>
<td>water-nsq</td>
<td>3.1</td>
<td>2</td>
<td>1560</td>
<td>1866</td>
<td>11.61  11.88 0.32</td>
<td>304/188 733/347 2/1</td>
<td>2.99</td>
</tr>
<tr>
<td>water-spa</td>
<td>3.66</td>
<td>2</td>
<td>1726</td>
<td>2092</td>
<td>14.19  14.83 0.38</td>
<td>309/194 798/371 2/1</td>
<td>3.72</td>
</tr>
<tr>
<td>radiosity</td>
<td>24.52</td>
<td>2</td>
<td>160712</td>
<td>183679</td>
<td>OOT   OOT 4693.05</td>
<td>OOT OOT 2/1</td>
<td>28124.97</td>
</tr>
</tbody>
</table>

Node visiting number reflects how many iterations were involved in each approach.
Outline

• Background
• Problems Formulation
• Solutions: design and implementation
• Evaluation
• Conclusion
Conclusion

- We addressed the two most severe efficiency problems in the original state-of-art IDFB MHP approach in this paper:
  - **Non-topological** node processing order which leads to repeated computation
  - **Eagerly update** of MHP information caused by symmetry of MHP analysis leads to redundant computation

**YES! We can make it faster!**

- We verified the efficiency improvement by significant experiments:
  - Our approach has a relative speed-up of $29.02 \times$ comparing to the original approach
  - Our approach also has a relative speed up of $10.00 \times$ comparing to the state-of-art non-IDFB approach
  - An order of magnitude improvement on efficiency