Static Conflict Analysis for Multi-Threaded Object Oriented Programs

Christoph von Praun
and Thomas Gross

Presented by Andrew Tjang
Authors

• Von Praun
  – Recent PhD
  – Currently at IBM (Yorktown Heights)
  – Compilers and runtime systems for OOP

• Gross
  – Stanford Graduate
  – CMU Faculty
  – Compilers, software construction
Introduction

- How to get info about object use/sharing in multi threaded environments?
- Introducing Object Use Graphs (OUG)
- Provides info to compiler for optimizations
  – o/w compiler makes assumptions
- Check for race conditions in programs (for sw engineers)
- Analysis done at compile time!
Background and Problems

- Escape analysis produce Heap Shape Graphs (HSG)
  - Classifies objects according to properties
- Info valid for *whole* program
  - Accesses in 2 threads cause obj to be shared
- How to get finer granularity?
OUGs

- OUG constructed to determine the “happened before” relationship
- Refines HSG’s escape info
  - Recognizes structural, temporal, and lock protections
- OUG foundation to concurrence aware compiler systems
- Distinguish effects of different threads on abstract objects
Terminology

• Abstract thread, abstract object
  – Compiler entities
  – Conservative approximate runtime entities

• Runtime thread, runtime object
  – Actual objects and threads that exist in a program’s execution
Example HSG

class Shared {
    int i;
    Shared () { i = 0; } // (20)
}

class Example extends Thread {
    static Shared s_field;
    static Object lock_ = new Object();

    static void main(String[] args) {
        Shared s_local = new Shared(); // (2), (3)
        s_local.i++; // (4), (6)

        s_field = s_local; // (6)
        s_field.i++; // (7), (8), (9)

        Thread t = new Example();
        t.start(); // (10)

        synchronized(lock_) {
            s_field.i++; // (11), (12), (13)
        }

        t.join(); // (14)
        s_field.i++; // (15), (16), (17)
    }

    void run() { // (22)
        synchronized(lock_) {
            s_field.i++; // (23), (24), (25)
        }
    }
}

Figure 2: HSG for the example program.
Modeling Object Uses

- Nodes in OUG = events
- Edges = “happened before” relation
- Nodes:
  - Get/Put, Load/Store/Escape, Tstart/Tjoin, Entry/Exit, Call
- Nodes have attributes
  - Abstract thread, program site, host object, accessed field, set of locks, etc...
OUG Edges

- Control-flow ordering
  - Order of events in threads (can be cyclic)
- Reference flow ordering
  - Restriction – object used after object creation or sharing
- Thread-relation ordering
  - Edges to/from TStart/Tjoin nodes
Conflicting vs. Safe

- Objects are conflicting if:
  - No ordering between events
  - Events from different threads
  - At least one event is a Put
  - Accesses are not done under locks

- Safe otherwise

- OUGs may have false positives
OUG Construction

- Determine abstract threads
- Build HSG
- Build OUGs from symbolic execution of abstract threads
- Analyze OUG for conflicting events
Determining Abstract Threads

- Represented by: \( T := (\text{tid}, (m_0\ldots m_n), \text{kind}, \text{multi}) \)
- Tid = unique id for thread
- M0-mn = entry methods for thread
- Kind = (init, main, or user)
- Mult = unique vs multiple
- Threads characterized by methods executed from call graph rooted @ entry methods
Computing the HSG

- Flow insensitive of data and reference relations
- Compositional
  - Methods are analyzed independently, and summaries used
- Summary has context (parameters return values, etc)
  - Reference vars id’d by alias set
    - AS:=(fieldmap, props, tidmask)
    - Fieldmap = field names to alias set (reachable)
    - Props=properties
    - Tidmask = abstract threads that access
More on HSGs

- Alias sets for class & abstract threads become root nodes
- Reachable from root (transitively) = global => global, could be multi thread accessible
- Method summary
  - $MS[m] = ((f_0...f_n), ret, except, alloc, reads, writes)$
  - $F_0...f_n = local\ variables$
  - Collection of alias sets
  - Abstract thread id noted at all object access sites
Symbolic Execution

- Narrows classification into conflicting and non conflicting
- This is where OUGs are constructed
- Maps onto the HSG
- Follows the program execution through the nodes (objects)
MOUGs

- Models relevant events at method level
- Control graph where actions that do not involve the object are pruned
- Created through single flow sensitive method traversal
- Relevant events are local variable l0-ln, or global alias set if class
class Shared {
    int i;
    Shared () { i = 0; } // (20)
}

class Example extends Thread {
    static Shared s_field;
    static Object lock_ = new Object();

    static void main(String[] args) {
        Shared s_local = new Shared(); // (2),(3)
        s_local.i++;
        // (4),(5)

        s_field = s_local;
        // (6)
        s_field.i++;
        // (7),(8),(9)

        Thread t = new Example();
        t.start(); // (10)

        synchronized(lock_) {
            s_field.i++;
            // (11),(12),(13)
        }

        t.join(); // (14)
        s_field.i++;
        // (15),(16),(17)
    }

    void run() {
        synchronized(lock_) {
            s_field.i++;
            // (22), (23), (25)
        } // (26)
    }
}

Figure 5: Example MOUGs.
Optimization

- @ Compile time, all execution paths = expensive traversals
- Save some time by avoiding equivalent method, thread, and locking context
  - Use site context $SC:=(m,(a_0\ldots a_n),\text{tid},\text{lockset})$
- Avoid descent into methods that have no effect on shared data
Conflict Analysis

• Events between new and escape safe
• Events before TSTART are safe if issued by a unique thread?
  – Unless thread is started multiple times
• If only Get events left, obj is read-only
• Else check if lock protected
• Else check readonlyness/lock protected of individual fields
## Benchmarks

<table>
<thead>
<tr>
<th>program characteristics</th>
<th>philo</th>
<th>elevator</th>
<th>mtrt</th>
<th>sor</th>
<th>tsp</th>
<th>hedd</th>
<th>mold</th>
<th>ray</th>
<th>monte</th>
</tr>
</thead>
<tbody>
<tr>
<td>appl loc</td>
<td>81</td>
<td>528</td>
<td>11298</td>
<td>300</td>
<td>706</td>
<td>28299</td>
<td>1402</td>
<td>1972</td>
<td>3674</td>
</tr>
<tr>
<td>appl classes</td>
<td>2</td>
<td>5</td>
<td>34</td>
<td>7</td>
<td>4</td>
<td>48</td>
<td>11</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>lib classes</td>
<td>129</td>
<td>142</td>
<td>158</td>
<td>132</td>
<td>141</td>
<td>208</td>
<td>129</td>
<td>131</td>
<td>146</td>
</tr>
<tr>
<td>methods in call graph</td>
<td>192</td>
<td>311</td>
<td>722</td>
<td>205</td>
<td>302</td>
<td>1025</td>
<td>224</td>
<td>270</td>
<td>441</td>
</tr>
<tr>
<td>bytecodes in call graph</td>
<td>3605</td>
<td>6820</td>
<td>20137</td>
<td>4483</td>
<td>6481</td>
<td>24375</td>
<td>6531</td>
<td>5982</td>
<td>8161</td>
</tr>
<tr>
<td>user threads</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>method spec</td>
<td>68</td>
<td>118</td>
<td>578</td>
<td>16</td>
<td>108</td>
<td>3653</td>
<td>111</td>
<td>150</td>
<td>267</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>compilation resources</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>shape analysis [s]</td>
<td>0.7</td>
<td>1.3</td>
<td>2.6</td>
<td>0.7</td>
<td>1.6</td>
<td>6.5</td>
<td>0.9</td>
<td>0.9</td>
<td>1.1</td>
</tr>
<tr>
<td>symb exec [s]</td>
<td>0.5</td>
<td>0.8</td>
<td>2.5</td>
<td>0.5</td>
<td>0.8</td>
<td>123.6</td>
<td>0.9</td>
<td>0.8</td>
<td>1.3</td>
</tr>
<tr>
<td>meth sites proc</td>
<td>103</td>
<td>191</td>
<td>1090</td>
<td>50</td>
<td>168</td>
<td>29254</td>
<td>156</td>
<td>209</td>
<td>431</td>
</tr>
<tr>
<td>meth sites reused</td>
<td>85</td>
<td>163</td>
<td>1640</td>
<td>43</td>
<td>123</td>
<td>60233</td>
<td>244</td>
<td>285</td>
<td>452</td>
</tr>
<tr>
<td>meth sites noeffect</td>
<td>100</td>
<td>179</td>
<td>855</td>
<td>81</td>
<td>163</td>
<td>29423</td>
<td>136</td>
<td>174</td>
<td>358</td>
</tr>
<tr>
<td>conflict analysis [s]</td>
<td>0.1</td>
<td>0.2</td>
<td>2.8</td>
<td>0.1</td>
<td>0.2</td>
<td>433.8</td>
<td>0.9</td>
<td>0.9</td>
<td>0.5</td>
</tr>
<tr>
<td>memory [MB]</td>
<td>0.5</td>
<td>3.0</td>
<td>14.7</td>
<td>0.5</td>
<td>1.5</td>
<td>263.5</td>
<td>1.5</td>
<td>3.4</td>
<td>3.8</td>
</tr>
</tbody>
</table>

*Table 1: Benchmark characterization and compilation properties.*
## Characteristics

<table>
<thead>
<tr>
<th>classification of HSG nodes</th>
<th>philo</th>
<th>elevator</th>
<th>mtrt</th>
<th>sor</th>
<th>tsp</th>
<th>heduc</th>
<th>mol</th>
<th>ray</th>
<th>monte</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>131</td>
<td>147</td>
<td>192</td>
<td>139</td>
<td>145</td>
<td>256</td>
<td>140</td>
<td>150</td>
<td>165</td>
</tr>
<tr>
<td>inst</td>
<td>43</td>
<td>65</td>
<td>199</td>
<td>44</td>
<td>62</td>
<td>467</td>
<td>51</td>
<td>71</td>
<td>79</td>
</tr>
<tr>
<td>inst unique</td>
<td>29</td>
<td>43</td>
<td>122</td>
<td>31</td>
<td>39</td>
<td>356</td>
<td>33</td>
<td>35</td>
<td>51</td>
</tr>
<tr>
<td>shared</td>
<td>10</td>
<td>13</td>
<td>97</td>
<td>3</td>
<td>13</td>
<td>184</td>
<td>16</td>
<td>29</td>
<td>36</td>
</tr>
<tr>
<td>shared readonly</td>
<td>3</td>
<td>6</td>
<td>55</td>
<td>1</td>
<td>6</td>
<td>116</td>
<td>6</td>
<td>12</td>
<td>28</td>
</tr>
<tr>
<td>shared lock-protected</td>
<td>6</td>
<td>3</td>
<td>36</td>
<td>1</td>
<td>4</td>
<td>30</td>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>shared mix-protected</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>shared conflicting</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>36</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OUGs</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes max</td>
<td>217</td>
<td>327</td>
<td>1618</td>
<td>286</td>
<td>311</td>
<td>83052</td>
<td>537</td>
<td>302</td>
<td>726</td>
</tr>
<tr>
<td>nodes median</td>
<td>50</td>
<td>95</td>
<td>74</td>
<td>116</td>
<td>99</td>
<td>417</td>
<td>99</td>
<td>64</td>
<td>59</td>
</tr>
<tr>
<td>edges max</td>
<td>435</td>
<td>689</td>
<td>6083</td>
<td>410</td>
<td>640</td>
<td>206456</td>
<td>616</td>
<td>616</td>
<td>2450</td>
</tr>
<tr>
<td>edges median</td>
<td>67</td>
<td>172</td>
<td>111</td>
<td>221</td>
<td>163</td>
<td>748</td>
<td>145</td>
<td>92</td>
<td>84</td>
</tr>
</tbody>
</table>

**Table 2:** Characteristics of HSG and OUGs (no arrays).
How well does it do?

<table>
<thead>
<tr>
<th></th>
<th>philo</th>
<th>elevator</th>
<th>mtrt</th>
<th>sor</th>
<th>tsp</th>
<th>hedc</th>
<th>mol</th>
<th>ray</th>
<th>monte</th>
</tr>
</thead>
<tbody>
<tr>
<td>global</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>abstract objects</td>
<td>13</td>
<td>38</td>
<td>91</td>
<td>14</td>
<td>30</td>
<td>201</td>
<td>25</td>
<td>39</td>
<td>54</td>
</tr>
<tr>
<td>allocation sites</td>
<td>18</td>
<td>55</td>
<td>117</td>
<td>18</td>
<td>42</td>
<td>256</td>
<td>27</td>
<td>56</td>
<td>59</td>
</tr>
<tr>
<td>access sites</td>
<td>135</td>
<td>526</td>
<td>1002</td>
<td>288</td>
<td>478</td>
<td>1954</td>
<td>963</td>
<td>466</td>
<td>399</td>
</tr>
<tr>
<td>r/w shared</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>abstract objects</td>
<td>7</td>
<td>10</td>
<td>59</td>
<td>5</td>
<td>9</td>
<td>107</td>
<td>14</td>
<td>24</td>
<td>20</td>
</tr>
<tr>
<td>allocation sites</td>
<td>12</td>
<td>17</td>
<td>89</td>
<td>4</td>
<td>13</td>
<td>180</td>
<td>19</td>
<td>43</td>
<td>29</td>
</tr>
<tr>
<td>access sites</td>
<td>111</td>
<td>246</td>
<td>956</td>
<td>197</td>
<td>337</td>
<td>1818</td>
<td>899</td>
<td>408</td>
<td>252</td>
</tr>
<tr>
<td>OUG (lock protection)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>abstract objects</td>
<td>2</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>76</td>
<td>8</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>allocation sites</td>
<td>2</td>
<td>9</td>
<td>19</td>
<td>4</td>
<td>5</td>
<td>163</td>
<td>7</td>
<td>31</td>
<td>29</td>
</tr>
<tr>
<td>access sites</td>
<td>21</td>
<td>168</td>
<td>165</td>
<td>155</td>
<td>190</td>
<td>1387</td>
<td>751</td>
<td>254</td>
<td>216</td>
</tr>
<tr>
<td>OUG (all)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>abstract objects</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>36</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>... improvement (%)</td>
<td>86</td>
<td>60</td>
<td>91</td>
<td>80</td>
<td>67</td>
<td>63</td>
<td>71</td>
<td>57</td>
<td>75</td>
</tr>
<tr>
<td>allocation sites</td>
<td>1</td>
<td>6</td>
<td>16</td>
<td>2</td>
<td>3</td>
<td>129</td>
<td>5</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>... improvement (%)</td>
<td>92</td>
<td>65</td>
<td>82</td>
<td>50</td>
<td>77</td>
<td>28</td>
<td>74</td>
<td>63</td>
<td>48</td>
</tr>
<tr>
<td>access sites</td>
<td>11</td>
<td>113</td>
<td>121</td>
<td>75</td>
<td>58</td>
<td>1110</td>
<td>529</td>
<td>144</td>
<td>118</td>
</tr>
<tr>
<td>... improvement (%)</td>
<td>90</td>
<td>54</td>
<td>87</td>
<td>62</td>
<td>83</td>
<td>38</td>
<td>41</td>
<td>65</td>
<td>53</td>
</tr>
<tr>
<td>avg/max alloc sites per obj.</td>
<td>1.0/1</td>
<td>1.5/2</td>
<td>4.0/9</td>
<td>2.0/2</td>
<td>1.0/1</td>
<td>4.1/63</td>
<td>1.3/2</td>
<td>2.0/4</td>
<td>3.8/9</td>
</tr>
<tr>
<td>conflicting fields</td>
<td>2</td>
<td>12</td>
<td>20</td>
<td>11</td>
<td>6</td>
<td>198</td>
<td>50</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>avg/max acc sites per field</td>
<td>5.5/8</td>
<td>9.3/29</td>
<td>5.7/23</td>
<td>6.8/11</td>
<td>9.7/14</td>
<td>4.8/33</td>
<td>10.6/127</td>
<td>5.5/13</td>
<td>5.9/23</td>
</tr>
</tbody>
</table>

**Table 3:** Static conflict detection (no arrays).
How useful?

<table>
<thead>
<tr>
<th></th>
<th>philo</th>
<th>elevator</th>
<th>mtrt</th>
<th>sor</th>
<th>tsp</th>
<th>hedc</th>
<th>mol</th>
<th>ray</th>
<th>monte</th>
</tr>
</thead>
<tbody>
<tr>
<td>shared</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>allocated</td>
<td>11</td>
<td>43</td>
<td>440</td>
<td>4</td>
<td>10011</td>
<td>861</td>
<td>2064</td>
<td>2103951</td>
<td>20020</td>
</tr>
<tr>
<td>actually shared</td>
<td>8</td>
<td>37</td>
<td>15</td>
<td>4</td>
<td>375</td>
<td>207</td>
<td>5</td>
<td>345</td>
<td>20013</td>
</tr>
<tr>
<td>conflict</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>allocated</td>
<td>2</td>
<td>33</td>
<td>6</td>
<td>2</td>
<td>5002</td>
<td>491</td>
<td>2051</td>
<td>2103667</td>
<td>20007</td>
</tr>
<tr>
<td>actually conflict</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>163</td>
<td>15</td>
<td>1</td>
<td>69</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4: Allocation of objects with their compile-time classification and the actual situation at runtime.
## How fast?

<table>
<thead>
<tr>
<th></th>
<th>mtrt</th>
<th>sor</th>
<th>tsp</th>
<th>mol</th>
<th>ray</th>
<th>monte</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>no instrumentation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>base</td>
<td>20.8</td>
<td>3.8</td>
<td>8.9</td>
<td>20.6</td>
<td>49.4</td>
<td>23.4</td>
</tr>
<tr>
<td>optimized(^1)</td>
<td>19.9</td>
<td>3.2</td>
<td>8.9</td>
<td></td>
<td>46.1</td>
<td>22.6</td>
</tr>
<tr>
<td><strong>object race checking</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stack-escape</td>
<td>41.7</td>
<td>3.9</td>
<td>23.8</td>
<td>64.0</td>
<td>116.1</td>
<td>41.5</td>
</tr>
<tr>
<td>global</td>
<td>29.6</td>
<td>3.9</td>
<td>23.8</td>
<td>65.5</td>
<td>111.5</td>
<td>41.3</td>
</tr>
<tr>
<td>shared r/w</td>
<td>29.0</td>
<td>3.9</td>
<td>23.9</td>
<td>65.4</td>
<td>110.9</td>
<td>42.0</td>
</tr>
<tr>
<td>OUG</td>
<td>28.5</td>
<td>3.9</td>
<td>10.3</td>
<td>66.0</td>
<td>82.7</td>
<td>40.9 (75%)</td>
</tr>
<tr>
<td>OUG optimized</td>
<td>27.7</td>
<td>3.3</td>
<td>10.1</td>
<td>38.4</td>
<td>73.0</td>
<td>40.4 (79%)</td>
</tr>
</tbody>
</table>

Table 7: Runtime in seconds and overhead of the program instrumentation (array access not instrumented).
Limitations

-Initializer and finalizer threads not considered
-Whole Program Knowledge
  - Reflection/dynamic class loading
-Naïve (conservative) thread ordering assumptions
Conclusion

- OUGs can provide finer grained analysis of shared objects
- Fewer accesses are classified as conflicting (incorrectly)
- OUGs solid foundation for reporting access conflicts and optimize synchronization operation placement