Subgraph Isomorphism

Aaron Blankstein, Matthew Goldstein

MIT 6.884

May 6, 2010
Subgraph Isomorphism

- Given graphs $G, H$
- Determine if $H$ is isomorphic to a subgraph of $G$
<table>
<thead>
<tr>
<th>Problem</th>
<th>VFLib</th>
<th>Simple Solution</th>
<th>Solutions</th>
<th>Analysis</th>
</tr>
</thead>
</table>

- **Problem:**
  - Diagram 1: Node B, C, B, A
  - Diagram 2: Node A, B, C, B, A

- **VFLib:**
  - Diagram 1: Node B
  - Diagram 2: Node A

- **Simple Solution:**
  - Diagram 1: Node C
  - Diagram 2: Node B

- **Solutions:**
  - Diagram 1: Nodes B, A
  - Diagram 2: Nodes B, A

- **Analysis:**
  - Diagram 1: Nodes C, B, A
  - Diagram 2: Nodes B, C, B, A
Outline

- Parallelized Subgraph Isomorphism Library
- Many failed methods
- Near-linear Speedup on Random Graphs
- Reasonable Parallelism on *Hard* Parasitic Graphs
VFLib

- General Fast Subgraph Isomorphism Library
- Memory efficient implementation of Ullman Algorithm
  - in-set, out-set
- Serial implementation only
- Works with large database of graphs
VFLib’s Algorithm

- Ullman Heuristics
- Optimized Data Structures
  - Matched Set for current match
  - in-set Array
  - out-set Array
  - Indexed by node-id
  - Stores step number (basically a log)

```
      5 1 2 7
```
Parallel Algorithm

- Add cilk_spawn calls for each graph child
- Deep copy of data structure whenever we spawn
- Massive memory overhead: solve by “coarsening”
Random Graph Results

- Good parallelism on random graphs
- Low burden: 312.66 / 314.71
Trouble in Parallel Paradise

- Coarsening solves our memory issue
- Not hard to imagine parasitic graphs
- 2D Meshes - Speedup: 1.001
Initial Strategies

- 2 Basic Approaches
- Spawn more
  - Faster Deep Copying
- Spawn better
  - Develop heuristics for where and when to cilk_spawn
Data Structure Solutions

- Cost values are in Seconds
- Scaled by times called per deep copy – next pair is x20 and backtrack is x0.6

<table>
<thead>
<tr>
<th></th>
<th>Deep Copy</th>
<th>Get-Next-Pair</th>
<th>BackTrack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrays</td>
<td>$6.7 \cdot 10^{-6}$</td>
<td>$1.2 \cdot 10^{-5}$</td>
<td>$1.1 \cdot 10^{-6}$</td>
</tr>
<tr>
<td>Mapsets</td>
<td>$8.8 \cdot 10^{-6}$</td>
<td>$1.2 \cdot 10^{-4}$</td>
<td>$1.8 \cdot 10^{-5}$</td>
</tr>
<tr>
<td>Bitsets</td>
<td>$5.1 \cdot 10^{-6}$</td>
<td>$3.2 \cdot 10^{-5}$</td>
<td>$3.0 \cdot 10^{-6}$</td>
</tr>
</tbody>
</table>
Spawn Heuristic Solutions

- Non-linear cutoff
- Mid-tree respawn
- Spawn first child
- Really just want to spawn when there's more work
Conditional Copy

- Deep copy only when a steal occurs – requires a Snapshot

```cpp
bool running = false
cleanCopy = state->deepClone()
for (p in state->nextPair())
    needs_clean = ! running
    if (running)
        nextState = cleanCopy->deepClone()
        nextState -> addPair()
    else
        running = true
        nextState = state
        nextState -> addPair()
    cilk_spawn match(nextState, &running, needs_clean)
    ....

if (needs_clean)
    nextState -> backTrack()
    * parentRunFlag = false
```
Parallelism

- Increased burden, but fixed most parasitic cases
Random Graphs Speedup

![Random Graph Speedup Plot](image-url)
2D Meshes Speedup

![Graph showing 2D mesh speedup](image)
Parallel Programming Difficulty

- **Main issue:** spawn cost $>>$ scheduling overhead
- Better spawn heuristics with active_workers()
- Our conditional copy is on the bleeding edge of *working* and *not working*
  - Possible language feature
  - Data Structure Wrapper - splitter
Future Work

- Optimizing for non-steal case. Mix log and snapshots.
- Design and implement a splitter hyperobject
Contributions

- Compared performance of several data structures for subgraph isomorphism
- Implemented a faster than current state-of-the-art subgraph isomorphism match algorithm.
- Detailed a general approach to dealing with large data structure copying for spawns in cilk platform
- Speculated on useful language features to enable conditional copying in cilk