HipG: Parallel Processing of Large-Scale Graphs

Article by:
E. Krepska, T. Kielmann, W. Fokkink, H. Ball

Appeared in:
SIGOPS Operating Systems Review, vol. 45, Issue 2, pg. 3-13

My information:
Patrick van Vliet, pvt240@few.vu.nl
Outline

• Large-Scale Graphs
• Challenges in graph processing
• What’s already out there?
• HipG in a nutshell
• What HipG doesn’t do for you
• What HipG does do for you
Outline

• More detailed look
• Achievements
• Conclusion
Large-Scale Graphs

• What are large-scale graphs?
  – Graphs
  – Large-Scale
    • OpenStreetMap (1.0 billion nodes, 2008)
    • China Telephone Networks (1.1 billion nodes, 2010)
  – Sparse
    • Facebook
    • Google
Challenges in graph processing

• Memory
• Partitioning
  – Load balancing
• Parallelizing
What’s already out there?

• Boost Graph Library
• Parallel Boost Graph Library
• ...

Patrick van Vliet, pvt240@few.vu.nl
HipG in a nutshell

Hierarchical Parallel Graph Algorithms

• Unified interface
• High-Level Framework
  – Abstraction
What HipG doesn’t do for you

• Ordering
• Handle fault tolerance
What HipG does for you

• Simplify coding
• Build graphs
• Partitions graphs
• Communication handling
  – Unified node access
• Global results
More detailed look

• Java
  – Portability
  – Efficiency
  – Ibis
• Locking
• Nodes
• Synchronizers
  – Divide-and-Conquer
More detailed look

```java
interface MyNode extends Node {
    public void found(SSSP sp, int d);
}

class MyLocalNode extends LocalNode<MyNode> implements MyNode {
    int dist = -1;
    public void found(SSSP sp, int d) {
        if (dist < 0) {
            dist = d;
            sp.Q.add(this);
        }
    }
    public void found0(SSSP sp, int d) {
        for (MyNode n : neighbors())
            n.found(sp, d);
    }
}

class SSSP extends Synchronizer {
    Queue<MyLocalNode> Q = new Queue<MyLocalNode>();
    int localQsize;
    public SSSP(MyLocalNode pivot) {
        if (pivot != null)
            Q.add(pivot);
        localQsize = Q.size();
    }
    @Reduce
    public int GlobalQSize(int s) {
        return s + Q.size();
    }
    public void run() {
        int depth = 0;
        do {
            for (int i = 0; i < localQsize; i++)
                Q.pop().found0(this, depth);
            depth++;
            barrier();
            localQsize = Q.size();
        } while (GlobalQSize(0) > 0);
    }
}
```

Fig. 3. Single-source shortest paths (breadth-first search) implemented in HipG.
Achievements

Figure 1: Reachability search from pivot $p$. 
Achievements

Figure 9: Speedup of Visitor, BFS and SpinJadi.

<table>
<thead>
<tr>
<th>Workers</th>
<th>Perfect speedup</th>
<th>Visitor</th>
<th>BFS/LN</th>
<th>SpinJadi-E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>80</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
</tr>
</tbody>
</table>
Achievements

![Bar chart showing performance comparison between different configurations.](image-url)
Conclusion

• High-Level -> Easy to code
• Sequential -> Parallel algorithms
• Partitioned graphs

Any questions?