Towards Approximate Computing: Programming with Relaxed Synchronization

Lakshminarayanan Renganarayana  
Vijayalakshmi Srinivasan  
Ravi Nair (presenting)  
Dan Prener  
*IBM T.J. Watson Research Center*

October 21, 2012
Emerging Use of Computing

- Emerging important applications
  - Search
    - Inexact answers to inexact queries
  - Proactive delivery of information
    - Advertisements, offers, news, alerts, advise
  - Media
    - Approximate rendition of audio, video, or images
  - Stream Processing
    - Fast decisions based on possibly incomplete data

- These applications benefit in cost-performance through the use of prediction techniques
- They allow trading off accuracy of results for speed of delivering results
Approximate Computing

- Current techniques
  - Exact (hence energy-inefficient) computation done on exact (hence expensive) hardware to produce results that need not be exact
  - With increasing unreliability and variability of devices in new generations of technology, the cost of such computation will increase dramatically

- What is needed: Approximate Computing
  - Computational models that allow computational results to fall in some acceptable range, and
  - Hardware that produces results that fall in some acceptable range
Dimensions of Approximate Computing

- **Data**
  - Reliable
  - Accurate
  - Less Accurate, less up-to-date, possibly corrupted

- **Hardware**
  - Variable
  - Less Exact

- **Computation**
  - Exact
  - Computing model today
Improving Efficiency through Approximation

- Data: Less Accurate, less up-to-date, possibly corrupted
- Hardware: Reliable, Accurate, Exact, Less Exact
- Computation: Path to Efficiency, Variable, Future model, Human Brain

Computing model today
Relaxed Synchronization

- Less Accurate, less up-to-date, possibly corrupted
- Reliable
- Accurate
- Exact
- Computing model today
- Relaxed Synchronization
- Path to Efficiency
- Variable
- Human Brain

Data

Hardware

Computation

Relaxed Synchronization
Relaxing Synchronization: Challenges and Opportunities

**Principal uses of synchronization**
- Ensuring that all threads see consistent values of shared variables (e.g. atomic updates with locks)
  - Best candidate for relaxing synchronization
- Ensuring that threads reach various points in their execution in a predictable manner (e.g. barrier)
  - Can relax (depending on context)
- Parallel update of data structures (e.g. linked lists)
  - Difficult to relax (can lead to fatal crash of the program)

**Key questions about relaxed synchronization**
- Will the program produce results that are acceptable?
- If acceptable, would the results be produced in considerably less time?
Kmeans Clustering

- Partition data into $k$ clusters

  - Randomly select $K$ “initial” means
  - Create $k$ clusters by associating every data point with the nearest centroid
  - Recompute centroids of $k$ clusters

- Iterate until centroids do not change

- Widely used
  - Market segmentation, data mining, computer vision, astronomy
Kmeans Synchronization Overhead

- Kmeans: 8 clusters with large input set
- OpenMP based parallel code
- Parallel execution on an IBM Power7 machine

Synchronization time as high as 90% for 8 threads
Relaxing Synchronization in Kmeans (NU-MineBench)

- Parallel evaluation of all points moving between clusters
  - Synchronization is used to update list of points moving
- Stopping criterion:
  } while (delta > threshold && loop++ < 500);
  - Number of points moving (delta) < 0.1% of total points (threshold)
  - Subject to a maximum of 500 iterations
- This is already an approximation
- What would happen if synchronization were relaxed?

![Performance for 8 Clusters](image)

- Execution Time (secs)
- Number of Threads
- Speedup

<table>
<thead>
<tr>
<th>Number of Threads</th>
<th>Original</th>
<th>Relaxed</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>60</td>
<td>60</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>40</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>30</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
</tbody>
</table>

© 2012 IBM Corporation
Performance Improvement

Significant speedup observed in many combinations. No combination showed a degradation in performance.
Quality of Solution

- Computed total sum of distances of all points from the centroids of clusters they belong to
  - Good indicator of quality of solution
- Each bar in graph represents a different relaxed run
  - Non-determinism is a side-effect of relaxation

Quality of solution is not degraded due to relaxed synchronization
Graph500

- Breadth-first search (BFS) of a very large scale-free graph
- Representative of certain real-life analytics problems
  - E.g. Mail a flyer to all people who are connected directly and transitively to a certain person or sets of persons
- Experiments performed on version of award-winning IBM solution

Sample Scale-Free Graphs
Initial Experiment: Relaxing Synchronization

- The parallel examination of nodes on a wavefront requires an atomic update of a data structure representing list of visited nodes
  - Atomic update is a synchronizing operation
- Question: What would happen if we updated the data structure with a simple OR, rather than an atomic-OR operation?
### Sample Result

<table>
<thead>
<tr>
<th>CPUS</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>THREADS</td>
<td>16</td>
</tr>
<tr>
<td>SCALE</td>
<td>20</td>
</tr>
<tr>
<td>Time taken with accurate sync</td>
<td>0.248</td>
</tr>
<tr>
<td>Time taken with relaxed sync</td>
<td>0.077</td>
</tr>
<tr>
<td>Performance gain with relaxed sync</td>
<td>3.20</td>
</tr>
<tr>
<td>Vertices visited in accurate list</td>
<td>646056</td>
</tr>
<tr>
<td>Vertices visited in relaxed list</td>
<td>645985</td>
</tr>
<tr>
<td>Vertices common in both</td>
<td>645985</td>
</tr>
<tr>
<td>Missing vertices</td>
<td>29</td>
</tr>
<tr>
<td>Other vertices with wrong level numbers</td>
<td>71</td>
</tr>
</tbody>
</table>

- Each column represents a different relaxed run
  - Relaxed results are non-deterministic
  - Very few vertices missing

- Results
  - Very few vertices with wrong level numbers
  - Better than 3x the performance

- In most cases, it was possible to detect all missing vertices by simply validating the obtained tree
Relax-and-Check: A Framework to Exploit Relaxed Synchronization

- Assume there is a criterion that allows us to estimate the quality of a solution without adding high cost
- Relax-and-Check
  - Modifies an existing program
    - Run initially without synchronizations
    - Revert to fully synchronized version if quality does not fall within acceptable limits
- Such criteria are often already explicit in programs
  - E.g. convergence criteria
- Other “syndromes” can often be identified

```c
result = f_original(input);

\[ \rightarrow \]

result = f_relaxed(input);
if (qualityNotAcceptable(result)) {
    //compute result using the original version
    result = f_original(input);
}
```
Sample Results

- Three benchmarks from STAMP suite
- Acceptability criteria of results identical to original
- Significant reduction of run time due to relaxation of synchronization requirement
Relax-and-Check Strategy for BFS: A Slight Twist

- Very high correlation between validation errors and time to solution
  - Possibly because of redundant visit of vertices
  - Also, these occurred principally when hardware parameters were not suitable for problem size

- Use validation errors as measure of quality
  - Validate the solution produced by approximate technique
  - If number of errors exceeds a threshold, say 1000, revert to synchronized solution for future instances with similar problem size

Relax-and-Check provides a technique to exploit the advantages of relaxed synchronization with a quality safety-net
## Applications Suitable for Relax-and-Check

<table>
<thead>
<tr>
<th>Domain / Applications</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scientific</td>
<td>Finite Difference Methods, Successive over-relaxation, N-body methods</td>
</tr>
<tr>
<td>Data Mining, Text Mining, Risk Analysis, Machine Learning</td>
<td>K-means, Hierarchical clustering, Multinomial regression, Feature reduction</td>
</tr>
<tr>
<td>Numerical optimization, Forecasting, Financial Modeling, and Mathematical Programming</td>
<td>Non-convex optimization solvers, constrained optimization, multi-dimensional cubing</td>
</tr>
<tr>
<td>EDA</td>
<td>VLSI place and route</td>
</tr>
<tr>
<td>Graph processing and Social Network Analysis</td>
<td>Mesh refinement, Metis style graph partitioning, Graph traversals</td>
</tr>
<tr>
<td>Program analyses in compilers and software engineering</td>
<td>Iterative data flow analysis</td>
</tr>
</tbody>
</table>

*An large majority of applications in these domains choose a good solution from many correct solutions, and use acceptance criteria such as convergence tests, fixed-point tests, or boolean acceptability tests*
Move Towards Attributes of the Human Brain

Data

Hardware

Computation

Less Accurate, less up-to-date, possibly corrupted

Reliable

Accurate

Exact

Computing model today

Path to Efficiency

Variable

Future model

Human Brain

© 2012 IBM Corporation