Detecting Task Phases from Power Traces

Joseph Granados, Jake Probst, Nick Armour, Jeffrey Bahns, Suzanne Rivoire
Sonoma State University

Chung-Hsing Hsu
Oak Ridge National Laboratory

Workshop on Performance Modeling, Benchmarking, and Simulation @ SC
November 13, 2016
Prior work

We can identify applications based on power traces
Task Type Recognition

• Collapse each trace into a vector of statistical features

• Use classifier to guess best matching task type

• We leverage existing classifier based on random forest of decision trees (accuracy 85-90%)

[Combs et al., E2SC 2014]
Limitations

• Operates on entire traces, with no insight into local behavior
• Can’t recognize novel combinations of known task types
• Doesn’t allow resource management policies to dynamically adapt to finer-grained phases of a job
• **Goal:** automatically partition a trace into concatenated *phases* and recognize the task type of each
Steps in Phase Recognition

1. Identify **change points** in power trace
   Ex: $t = \{20, 150, 300, 430\}$

2. Identify intervals as **candidate phases**
   Ex: $[0, 20); [20, 150); [0, 150)$…

3. **Predict the task type** of each candidate phase
   Ex: $[0, 20): idle; [20, 150): FFT…$

4. Choose the best **final partition** of the trace
   $[0, 150): FFT$
   $[150, 300): sort$
   $[300, 430): GUPS$
   $[430, end): idle$
Experimental Setup

- Dataset of 388 traces from 21 “kernels”
  - **NPB**: bt, cg, ft, lu, sp, ua
  - **Mahout data analytics**: ALS, bayes, SGD, kmeans
  - **SystemBurn**: Tilt, fft1d, fft2d, dgemm, gups, scublas
  - **Other**: Nsort (external sort), primes95, STREAM, graph500, baseline (idle)
- Iteratively and randomly:
  - Remove 5 traces from dataset and concatenate to form a “test trace”
  - Build random forest from remaining traces
  - Partition test trace into kernels
- Correctness metric: how many data points in the trace were assigned to the right kernel?
Change Point Detection

• **Definition**: detecting abrupt changes in the statistical properties of a time series

• **Hypothesis**: Since we’ve shown that different task types have different statistical properties, the boundaries between different task types should also be *change points*

• **Goal**: detect a superset of the actual phase boundaries
  – We can weed out spurious change points in later steps…
  – …at the cost of computational complexity
Change Point Detection Algorithm

• Evaluated variants of *binary segmentation* [Scott and Knott, 1974]

• Basic idea:
  – Find best single changepoint in dataset; stop if none found
  – Recursively use to partition dataset and repeat

• Our best variant: *wild binary segmentation (WBS)* [Fryzlewicz, 2014]
  – Search for “best changepoint” in random intervals of different lengths
  – Better for irregularly spaced / short phases
Change Point Detection Example

Change Point Detection in Trace with 7 Phases

Watts

Seconds
Change Point Detection Results

• “Correct” change point: within 3 samples of actual task type transition

100 Rounds of Change Point Evaluation
Phase Count: 5 | Average Total Length: 51.98 min

Recall

Precision
Candidate Phase Identification

- Identify pairs of change points as *candidate phases*.
- **Minimal approach:** consecutive pairs only
  - Computationally simplest
  - …but will always fail to recognize internally complex task types
- **Maximal approach:** all possible pairs
  - Computationally expensive
  - …but guarantees inclusion of all real phases if change point algorithm worked
- **Our approach:** maximal (computationally tractable for our traces)
Final Partition

- **Build graph**: nodes for change points, edges for candidate phases

- **Weight edges** based on confidence of task type prediction [see next slide]

- **Compute longest path** to get final partition
**Edge Weights**

- Use internal properties of random forest
- **Certainty**: what fraction of trees voted for this task type?
- **Proximity**: how similar is this phase’s path through the trees to the paths taken by others in its type?
- **Weight by** interval length

$$0.5 \times (\text{certainty} + \text{proximity to traces of predicted type}) \times \text{interval\_length}$$
Examples
Correctness: Histogram

- Metric: number of data points attributed to the right “kernel” over 300 runs
Correctness throughout the length of the trace

Percentage correct at given points in a trace.
Conclusions

• Can break a trace into its constituent phases with high accuracy (mean 78%)

• Possible improvement
  • Prune candidate phases to reduce computational complexity
  • Use internal measures of trace complexity to tune target number of change points
  • Explore other methods of computing edge weights
  • Try higher frequency power measurements (RAPL).

• Possible extensions
  • Adapt to mix of known and unknown task types
  • Online recognition
Questions?
# Test Machines (Single-Node)

<table>
<thead>
<tr>
<th></th>
<th>LC</th>
<th>RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Core i5-750 @2.67Ghz</td>
<td>Intel Core i7-3770 @3.40Ghz</td>
</tr>
<tr>
<td>RAM</td>
<td>8GB</td>
<td>8GB</td>
</tr>
<tr>
<td>GPU</td>
<td>GeForce GTX 650 Ti 1GB</td>
<td>GeForce GTX 670 2GB</td>
</tr>
<tr>
<td>Power</td>
<td>85-252W</td>
<td>74-309W</td>
</tr>
</tbody>
</table>
Random Forest Feature Vector

- Normalized Max
- Normalized Min
- Standard Deviation
- Skewness
- Kurtosis
- Serial Correlation
- Nonlinearity
- Self-similarity
- Chaos
- Trend
- Skewness of detrended trace
- Kurtosis of detrended trace
- Serial Correlation of detrended trace
- Nonlinearity of detrended trace
- 4 Fourier Coefficients, skipping first
Trace Complexity

- We define the complexity of a single trace as $\frac{\log(\text{number_of_change_points})}{\log(\text{trace_length})}$
- Different thresholds can be used to determine the change in power required to define a single change point.
  - Based on Standard Deviation
  - Based on range
  - Based on Interquartile Range
  - Based on mean absolute deviation