Beacons: An End-to-End Compiler Framework for Predicting and Utilizing Dynamic Loop Characteristics
Girish Mururu*, Sharjeel Khan*, Bodhisatwa Chatterjee*, Chao Chen, Chris Porter, Ada Gavrilovska, Santosh Pande (* contributed equally to the paper)
School of Computer Science, Georgia Institute of Technology
OOPSLA’ 23, October 25-27, 2023 (Cascais, Portugal)

Introduction and Motivation
• In HPC environments, sharing of resources can badly affect performance if done wrong.
• No sharing leads to poor resource utilization of the systems and long queue wait times for users.
• State of the art resource managers in HPC environments currently estimate workload using three ways:
  • Application Profiling
  • History-based mechanism
  • Application Domain Knowledge
• These approaches are agnostic to the fact that workloads are input-dependent, and can fluctuate at different program phases.
• Current scheduling decisions suffer from detection and reaction lag, an error of 3%

Dynamic Trip Counts of Loops
• The sharing of resources requires the scheduler to determine dynamic cache interference between co-executing applications.
• Memory (Cache) footprint
• Duration of execution overlap
• These resources are determined by dynamic trip counts of loop.
• 55% loops are irregular or unanalyzable for trip counts (e.g. while loops, multi-exit, etc).
• Dynamic trip counts can be estimated through an ML-based model.

We present Beacons Framework, an end-to-end compiler and scheduling framework, that estimates dynamic loop characteristics, encapsulates them in compiler-instrumented Beacons in an application, and broadcasts them during application runtime, for proactive workload scheduling.

Beacon Scheduler
• Scheduler optimizes the sharing of the Last-Level Cache (LLC) among scheduled processes by changing between the two modes
  • Reuse Mode:
    • A reuse beacon for a loop will lead to a check if all current processes fit in the cache. If not, the process will be put in wait queue until a spot opens for it.
    • A streaming beacon for a loop is suspended until no more reuse processes are active.
• Streaming Mode:
  • Streaming Mode executes as many streaming processes without exceeding memory bandwidth
  • When we have many reuse processes in the waiting, we switch back to reuse mode.

Results

<table>
<thead>
<tr>
<th>Loop Characteristic</th>
<th>Estimation Technique</th>
<th>Evaluation Method</th>
<th>Utility in Workload Scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop Trip Count</td>
<td>Multi-Class Classifications Models (Supervised Machine Learning)</td>
<td>Decision Tree</td>
<td>To enhance the loop timing and loop memory footprint.</td>
</tr>
<tr>
<td>Loop Timing</td>
<td>Multi-Variable Linear Regression (Supervised Machine Learning)</td>
<td>Regression Equation</td>
<td>To anticipate how long an application will require the loop cache memory.</td>
</tr>
<tr>
<td>Loop Memory Footprint</td>
<td>Extension of Polyhedral Memory Analysis</td>
<td>Closed-form Formula</td>
<td>To determine the amount of cache memory required by the application’s loop nest.</td>
</tr>
<tr>
<td>Loop Data Reuse Behavior</td>
<td>Static Reuse Distance Analysis (LLVM Loop Cache Analysis)</td>
<td>Boolean Variable</td>
<td>To guide how different applications can be efficiently co-located without thrashing.</td>
</tr>
</tbody>
</table>

Loop Categorization Analysis
• Normally Bounded — Normal Exit (NBNE)
• Irregularly Bounded — Normal Exit (IBNE)
• Normally Bounded — Multi Exit (NBME)
• Irregularly Bounded — Multi Exit (IBME)

Loop Trip Count Estimation
• Compiler pass will backslice critical variables for loop termination to the pre-header of the loop.
• The set of its backslashed critical variables will serve as the feature-set while the trip count will be the output label in the decision tree training that gets inserted into the code.

Loop Timing Estimation
• Loop-nest timing is a function of the trip-counts of each loop in the loop nest.
• Any loop nest L with n inner-nested loops with individual trip-counts \(N_L, N_{L_1}, \ldots, N_{L_n}\) can be written as:
  \[T_L = \sum_{i=1}^{n} c_i x_i + \ldots + c_n x_n\]
• The constants are generated through regression on training runs.

Conclusion
• Our compiler analysis and machine learning techniques can accurately predict the loop trip count (average accuracy of 79.9%), the loop timing (average accuracy of 79.13%), and the loop memory footprint (average error of 3%).
• The Beacon scheduler shows an average throughput gain of 2.62x over a reactive scheduler called Merlin, and a gain of 1.9x over widely used Completely Fair Scheduler on 51 diverse benchmarks.
• Our framework based on dynamic loop characteristics can efficiently manage system resources to schedule processes in a HPC environment.