Modeling the Performance of GHOST – Geophysical High Order Suite for Turbulence

Ramanuja Simha

Mentor: Benjamin Mayer
Presentation Outline

- Motivation for the work
- Overview of Empirical and Analytic Performance Modeling
- Results – Empirical and Hybrid (Empirical + Analytical) models
- Conclusions and Future work
Why performance modeling?

- Predict the performance of an application on non-existent hardware.
- Analyze the performance of modifications to hardware setup.
- Compare the performance of multiple hardware configurations without the need to purchase or obtain access to desired system.
GHOST

✦ What is GHOST?
  • The Geophysical High Order Suite for Turbulence (GHOST) is a highly scalable Fortran 90-95 pseudo–spectral code that solves a variety of PDEs that are often encountered in turbulence studies.

✦ Why GHOST?
  • The GHOST application was chosen as the first application to model as it is one of the easier to produce a basic performance model for.
  • There is also the opportunity to build a full complex model to better handle leading edge machines and problem sizes.
What is performance modeling?

- Providing insight into performance issues – leads to better purchasing decisions.
- In this work, evaluating the feasibility of modeling the execution time of a software application.
- Predicting the run time based on software input parameters and hardware architecture.
Performance Modeling Process

Empirical Performance Model
- Acquire performance data from several systems.
- Use data mining approaches to reduce noise and build model.
- The model is built with statistical analysis and does not require in-depth examination of the application code or its functionality.
- Advantage: Relatively quick to build as there is no requirement for application knowledge.
- Disadvantages:
  a) This might not account for all the various application parameters affecting its performance.
  b) Can only evaluate system architectures similar to those for which we have data.
- Challenge: Acquiring data from many different systems.
Analytical Performance Model
- Examine the code and identify the scaling properties of computation and communication times.
- Single processor equation:
  \[ Time_{SingleProcessor} = Time_{Computation} + Time_{Memory} \]
- Multi-processor equation:
  \[ Time_{Multi-Processor} = (Time_{SingleProcessor} / np) + Time_{Communication} \]
- Advantages:
  a) Can account for more application related parameters, resulting in a more accurate model.
  b) Evaluate systems of radically different architectures.
- Disadvantage: Difficult to build and can take a very long time.
- Challenge: Understanding the parallel code accurately enough to build the performance model may require collaboration with the developers.
Empirical Performance Model

- We use the following data mining classifiers throughout the model building process – Linear Regression, Multilayer Perceptron (Neural Networks), and Support Vector Machines.

1) **Build Feature Set** (next slide)

2) **Noise identification in the data set**
   - Removing noisy data points helps to avoid overfitting and improves the overall prediction accuracy.
   - Data set does not require any pruning for noise (from Fig.1).
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vendor</td>
<td>The vendor name (AMD and Intel)</td>
</tr>
<tr>
<td>core</td>
<td>Core size (nm)</td>
</tr>
<tr>
<td>ntasks</td>
<td>Number of MPI tasks</td>
</tr>
<tr>
<td>tdp</td>
<td>Thermal design point of the CPU (W)</td>
</tr>
<tr>
<td>frequency</td>
<td>Frequency of the CPU measured, when data is recorded (MHz)</td>
</tr>
<tr>
<td>l1cache</td>
<td>L1 cache size (KB)</td>
</tr>
<tr>
<td>l1assoc</td>
<td>L1 cache associativity</td>
</tr>
<tr>
<td>L2cache</td>
<td>L2 cache size (KB)</td>
</tr>
<tr>
<td>l2assoc</td>
<td>L2 cache associativity</td>
</tr>
<tr>
<td>l3cache</td>
<td>L3 cache size (KB)</td>
</tr>
<tr>
<td>l3assoc</td>
<td>L3 cache associativity</td>
</tr>
<tr>
<td>peakmemspeed</td>
<td>Theoretical maximum memory speed using all sockets (KB/s)</td>
</tr>
<tr>
<td>maxmemspeed</td>
<td>Achieved maximum memory speed using all sockets (KB/s)</td>
</tr>
<tr>
<td>memefficiency</td>
<td>maxmemspeed / peakmemspeed</td>
</tr>
<tr>
<td>memtype</td>
<td>Memory type (DDR2 and DDR3)</td>
</tr>
<tr>
<td>memspeed</td>
<td>Memory speed per socket (MHz)</td>
</tr>
</tbody>
</table>
Empirical Performance Model

Fig. 1. Prediction error percentage (PEP) on training data - The ids of data points are on the x-axis and their prediction error percentage values are on the y-axis (log scale).
3) Feature selection

- Is a process that identifies a subset of features which allow the classifier to predict the class value faster and with high accuracy.
- Noisy features might adversely affect the overall prediction accuracy of the classifier.
- WrapperSubsetEval (attribute evaluator) along with BestFirst (search method).
4) Prediction results

- We conduct our experiments on various folds values so that the prediction results give a fair picture of the quality of the empirical model that has been built.
- Fig.2 shows the correlation coefficient between measured and prediction run times.
- On an average, the prediction errors of Linear Regression, Multilayer Perceptron, and Support Vector Machines are 41.59%, 4.95%, and 10.73%. This agrees well with other work in the field.
Empirical Performance Model

Fig. 2. Prediction - correlation coefficient. The number of folds used for the cross validation are on the x-axis and the correlation coefficient for the predicted values are on the y-axis.
1) Single Processor Model

- \( Time_{\text{SingleProcessor}} = Time_{\text{Computation}} + Time_{\text{Memory}} \)
- \( Time_{\text{Computation}} \) is predicted by modeling the FLOP count of individual subsections of the code
  
  a) \( Time_{\text{Computation}} = Time_{\text{Computation-FFTW}} + Time_{\text{Computation-GHOST}} \)
  
  b) \( Time_{\text{Computation-FFTW}} \) - We measure the FLOP count for the operations performed by the FFTW library
  
  c) Using this measure for different input sizes \( n \), a polynomial in \( n \) is fitted using the least squares method
  
  d) \( Time_{\text{Computation-FFTW}} = (O(n^2) \cdot \log(n))/(\text{H/W FLOP count} \cdot \text{Instructions per Cycle}) \)
  
  e) \( Time_{\text{Computation-GHOST}} \) - GHOST code analysis provides a direct upper bound of the FLOP count
Hybrid Performance Model

f) \( \text{Time}_{\text{Computation-GHOST}} = \mathcal{O}(n^3) / (\text{H/W FLOP count} \times \text{Instructions per Cycle}) \)

- \( \text{Time}_{\text{Memory-FFTW}} \) is modeled by taking into account the percentages of the overall runtime that is spent in doing the memory related operations
  
  a) \( \text{Time}_{\text{Memory}} = \text{Time}_{\text{Memory-FFTW}} + \text{Time}_{\text{Memory-GHOST}} \)
  
  b) \( \text{Time}_{\text{Memory-FFTW}} \) and \( \text{Time}_{\text{Memory-GHOST}} \) indicate the time spent by the respective parts of the code on memory related operations (memory create, memory copy, etc)
  
  c) Both \( \text{Time}_{\text{Memory-FFTW}} \) and \( \text{Time}_{\text{Memory-GHOST}} \) follow a \( \mathcal{O}(n^p) \) curve
Hybrid Performance Model

2) Multi-processor Model

- \( Time_{\text{Multi-Processor}} = (Time_{\text{SingleProcessor}} / \text{Number of Processors}) + Time_{\text{Communication}} \)
- \( Time_{\text{Communication}} \) includes latency
- \( Time_{\text{Communication}} \) is predicted by modeling the FFT (Fast Fourier Transform) communication across processors as this dominates the data transfer across processors
- For any \( n \) (input size) and \( np \) (processor count), the data transfer between processors can be modeled as:
  \[ (\text{chunk}_1 \times \text{count}_1 + \text{chunk}_2 \times \text{count}_2) \times \text{sizeof(datatype)} \]
Hybrid Performance Model

- $\text{chunk\_size1} = \frac{n^3}{(2 \times np^2)}$ and $\text{count1} = c \times f1(np)$, where $c = 3604$
- $\text{chunk\_size2} = n \times f2(np)$ and $\text{count2} = c \times (np/2)$, where $c = 3604$
- The total communication time for various chunk sizes can be measured by making use of tools such as MPIbench
- On average for input size $n = 256$, the error in the prediction is 8.16% (Fig.3)
Fig. 3. Performance Prediction (\(n=256\)). The number of processors \(np\) is on the x-axis and time (seconds) is on the y-axis.
Conclusion

- It is possible to rapidly produce an empirical model which gives good performance of architectures similar to what we built the model on.
- Analytic models take longer to build, but offer exact solutions for communication and computation performance for large portions of the model. This yields good accuracy in the ranges tested.
Future Work

- Verify the produced models on larger systems and larger data sizes.
- Increase the operational range over which the models work with the intent of capturing the performance trade off between MPI and OpenMP which current users are forced to use for large runs.
- The next generation of microprocessors and accelerators have unique and interesting architectures. Evaluate the performance of the current code on these processors and provide feedback to the application authors.
Acknowledgements

- Mentor: Benjamin Mayer
- CSS-Core: John Dennis, Jose Garcia, Michael Oberg, Henry Tufo, Jennifer Williamson, and Matthew Woitaszek
- GHOST Team: Pablo Mininni, Annick Pouquet, and Duane Rosenberg
- NCAR: Rich Loft and Kristin Mooney
- Work supported by NSF