Classifying Performance Bounds Using Machine Learning

Lewis Littman*
University of Bristol
Bristol, UK

Tom Deakin†
tom.deakin@bristol.ac.uk
University of Bristol
Bristol, UK

Abstract

Traditional performance analysis tools, such as the Roofline model, require visual interpretation to determine performance bounds. For CPUs which have complex cache hierarchies and front-end out-of-order capabilities—that is the CPUs we use for high performance computing—accurately identifying the true performance bound is challenging. This work is the first steps towards a data-driven approach to performance modelling, leveraging Machine Learning techniques. We build and evaluate a number of supervised and unsupervised models using a new curated data set of performance counters collected from well-understood (i.e., easily labeled) benchmark applications. We further analyse the data set and highlight potential "performance fingerprints" obtainable using this methodology.

CCS Concepts

• **Computing methodologies** → **Machine learning**; **Concurrent computing methodologies**; **Parallel computing methodologies**.

Keywords

performance analysis, performance counters, machine learning, high performance computing

1 Data set construction

A new data set has been created for this preliminary study into this methodology. It consists of performance counter data collected using the Linux `perf` tool whilst running benchmark applications. The performance counters are listed in Table 1 and benchmark applications in Table 2. The applications were selected to capture different Arithmetic Intensities (as defined by the Roofline Model [9]) and where their performance limiting factor was already well known, making labelling easy. We also confirmed the label through Roofline analysis. One limitation is there are few examples of floating-point bound applications; however, this is indicative of much current high-performance software where the majority of codes are bound by data throughput.

The data set was collected on the University of Bristol's BlueCrystal Phase 4, which uses two 14-core Intel Xeon E5-2680 v4 (Broadwell). This is an out-of-date processor, however the interpretation of performance counters is well understood, and more modern CPUs offer similar counters. Critically , the focus of this work is to evaluate the methodology of the machine learning approach—the choice of processor to collect the data set is arbitrary.

In order to collect all counters shown in Table 1, we run the application three times, collecting a subset and collecting into a single record (similar to multiplexing). The applications are parallelised with OpenMP, and executed with one thread pinned to each core. We collected 100 records for each application, with a total data set size of 1,200 records.[1]

Data preprocessing is a very important stage in the use of machine learning and can drastically affect the performance of a model [5]. For this data set, pre-processing allows us to extract more meaning in these values in order to characterise the performance limiting factor. For example, the number of instructions and number of clock cycles are not directly informative, but are useful in calculating metrics which are such as instructions per cycle. We pre-process the data and calculate: IPC (Instructions / cycles), Cache miss ratio (cache misses / cache references), and ratio of vectorised floating point instructions (packed instructions / instructions). Each record in the data set therefore contains: GFLOPs, FLOPc, IPC, retiring %, bad speculation %, frontend bound %, backend bound %, vectorised SP instruction ratio, vectorised DP instruction ratio, cache miss ratio, L1 cache miss ratio, L2 cache miss ratio and L3 cache miss ratio.

2 Machine Learned Models

As a baseline, we implemented a uniform random classifier which predicts "compute bound" or "bandwidth bound" with equal likelihood, along with a proportional model with probabilities for classes equal to the distribution in the data set, and a majority model which always predicts "bandwidth bound". The precision and recall are shown in Table 3, which they are little better than random.

We trained the following models: Decision Tree, k-Nearest Neighbours, Logistic Regression, Random Forest, Support Vector Machine, and a multi-layer perceptron.[2] We used a stratified data splitting to ensure that each set contains equal proportions of compute and bandwidth bound data. The data was standardised to avoid side effects of models which are sensitive to scale. Training these models with default parameters results in perfect accuracy, indicating over-fitting, or else easily separable data (see t-SNE analysis below). To mitigate over-fitting, we use leave-one-out cross-validation [10]. The accuracy of these models in Table 4 shows a clear improvement over the random models in all cases.

To further analyse the data set to explore issues such as trivial separability or low intraclass variance, we applied t-Distributed Stochastic Neighbour Embedding (t-SNE). This reduced-order visualisation of the data preserves all relationships between data (similar to Principal Component Analysis). Figure 1 shows this, where each cluster is one of the benchmark programs/problem sizes. This shows that the classes are distinctly separable, confirming the high accuracy classification for these classes. It also highlights the promise

*This poster follows his final-year thesis [3].
†Corresponding author.

[1]The dataset is available on Zenodo [4].
[2]The MLP had one hidden layer of 50 neurons with the ReLU activation function and trained with the Adam optimiser.

Table 1: Performance counters used for data set collection

Performance counters	Meaning
Instructions	number of instructions executed (used for normalisation)
Cycles	number of clock cycles during execution
Cache References	number of L1/L2/LLC references
Cache Misses	number of L1/L2/LLC misses
GFLOPs	number of floating-point operations per second
FLOPc	number of floating-point operations per clock cycle
Vectorised (packed) Instructions	number of vectorised floating-point instructions executed
Retiring	percentage of pipeline slots where useful instructions have been executed
Frontend bound	pecentage of time stalled waiting for instructions
Bad speculation	percentage of time wasted due to incorrect branch speculation
Backend bound	percentage of time stalled waiting for data

Table 2: Benchmark applications used for data set collection

Code	Category	Problem sizes
SGEMM	FP32 compute bound	15k-by-15k, 10k-by-10k matrix elements
DGEMM	FP64 compute bound	15k-by-15k, 10k-by-10k matrix elements
miniBUDE [7]	FP32 compute bound	128 poses per work-item, work-group size 1
STREAM [6]	Main memory bandwidth bound	10M FP64 elements
LBM D2Q9	LLC cache or main memory bandwidth bound	256-by-256 grid, 1024-by-1024 grid
HPCCG (MiniFE) [2]	Main memory bandwidth bound	50-by-50-by-50 grid
3D Heat stencil [8]	LLC bandwidth bound	120-by-120-by-120 grid
LU solver (Intel MKL)	LLC bandwidth bound	8192-by-8192, 16384-by-16384 matrix elements

Table 3: Baseline model results

Class	Uniform			Proportional			Majority		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Bandwidth Bound	0.56	0.42	0.48	0.64	0.59	0.62	0.58	1.00	0.74
Compute Bound	0.40	0.54	0.46	0.49	0.54	0.51	0.00	0.00	0.00
Accuracy		0.47			0.57			0.58	

Table 4: Machine Learned model results

Model	Average Accuracy
Decision Tree	0.833
k-NN	0.917
Logistic Regression	0.917
Random Forest	0.833
SVM	0.917
MLP	0.917

of this approach, where we can get much information from a performance counter data set to identify performance limiting factors, and we plan to increase the categories to include other bounds in future.

This analysis also highlights that these codes have some intrinsic "performance fingerprint" which would allow for easy identification by a model (perhaps explaining the findings of Antici, et al. [1]). We are pursuing this in further study.

References

[1] Francesco Antici, Andrea Bartolini, Zeynep Kiziltan, Ozalp Babaoglu, and Yuetsu Kodama. 2024. MCBound: An Online Framework to Characterize and Classify Memory/Compute-bound HPC Jobs. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15. doi:10.1109/SC41406.2024.00062

[2] Michael A. Heroux, Douglas W. Doerfler, Paul S. Crozier, James M. Willenbring, H. Carter Edwards, Alan Williams, Mahesh Rajan, Eric R. Keiter, Heidi K. Thornquist, and Robert W. Numrich. 2009. *Improving Performance via Mini-applications*. Technical Report. Sandia National Laboratories.

[3] Lewis Littman. 2025. Classifying Performance Bounds Using Machine Learning. Bachelor's thesis.

[4] Lewis Littman and Tom Deakin. 2025. Classifying Performance Bounds Using Machine Learning [Data set]. doi:10.5281/zenodo.17194638

[5] Kiran Maharana, Surajit Mondal, and Bhushankumar Nemade. 2022. A review: Data pre-processing and data augmentation techniques. *Global Transitions Proceedings* 3, 1 (2022), 91–99. doi:10.1016/j.gltp.2022.04.020 International Conference on Intelligent Engineering Approach(ICIEA-2022).

[6] John D. McCalpin. 1995. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on*

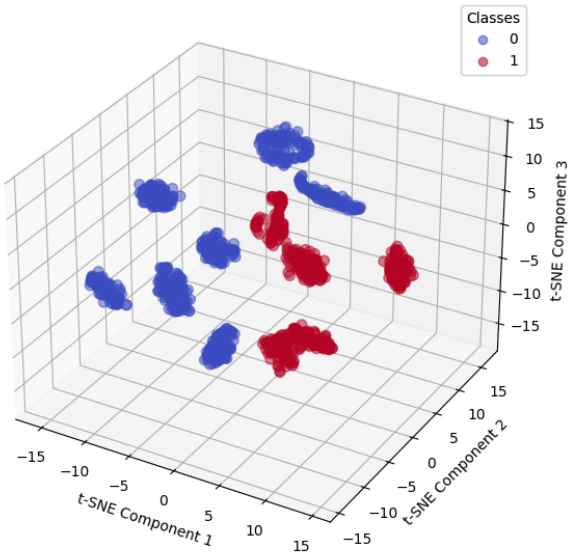

Figure 1: t-SNE analysis for the data set, where Class 0 represents bandwidth bound and Class 1 represents compute bound

Computer Architecture (TCCA) Newsletter (Dec. 1995), 19–25.

[7] Andrei Poenaru, Wei-Chen Lin, and Simon McIntosh-Smith. 2021. A Performance Analysis of Modern Parallel Programming Models Using a Compute-Bound Application. In *High Performance Computing*, Bradford L. Chamberlain, Ana-Lucia Varbanescu, Hatem Ltaief, and Piotr Luszczek (Eds.). Springer International Publishing, Cham, 332–350.

[8] Louis-Noel Pouchet and Tomofumi Yuki. 2016. PolyBench/C benchmark suite—version 4.2.1(beta). https://github.com/MatthiasJReisinger/PolyBenchC-4.2.1

[9] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (April 2009), 65–76. doi:10.1145/1498765.1498785

[10] Luke A. Yates, Zach Aandahl, Shane A. Richards, and Barry W. Brook. 2023. Cross validation for model selection: A review with examples from ecology. *Ecological Monographs* 93, 1 (2023), e1557. doi:10.1002/ecm.1557