

PBound: A Tool for Performance Bound Modeling

1 Introduction

PBound is a tool for estimating upper performance bounds of C/C++ applications through static compiler analysis. The tool generates parameterized expressions for different types of memory accesses and integer and floating-point computations. Combined with architectural information, upper bounds on the performance of an application on a particular system can be estimated. Application designers can test observed performance against these bounds to calculate the efficiency of their implementation where efficiency is defined to be the ratio of achieved performance to the performance bound. This enables more effective performance tuning than using theoretical peak performance. Alternately, if the achieved efficiency is high but the calculated bound is low, a different algorithm or architecture can be explored to improve the performance bounds. For example, for an iterative linear solver limited by memory bandwidth, performance could be improved by using a less bandwidth-intensive algorithm or by upgrading the memory.

Why is it important? Current methods to rate the implementation on a particular machine rely on calculating its efficiency w.r.t. the theoretical peak of the machine without considering whether it can be reached at all. An overestimated upperbound may cause a truly efficient implementation of a particular algorithm to be classified as an inefficient one leading to wasted development effort to improve it. Furthermore, two implementations with perceived similar low efficiency may have different true efficiencies. Using performance bounds sets a more accurate upper bound on performance.

Why no one else does it? Static analysis can at best conservatively estimate dynamic operations. Recursion, run time parameters and dynamic allocation skews the computed bounds. Further, competition for resources between applications cannot be modeled. However, many scientific codes are written as a series of loops that perform computation on arrays and can be analyzed accurately.

2 Implementation and Quantitative Evaluation

PBound is built on the ROSE Compiler framework. It traverses the SAGE Abstract Syntax Tree in a top-down and bottom-up manner and counts the bytes of data loaded and stored, as well as integer and floating point operations.

We ran PBound on `dgemm.c` an application in the `cblas` library and a sparse matrix vector multiplication application. PBound generates the bounds for each loop in the application. Figure XX shows a snippet of input code from `dgemm.c` and the resulting output code and computed bounds.

<pre>for (j = 1; j <= *n; ++j) { i__2 = *m; for (i = 1; i <= *m; ++i) { C(i,j) = 0.; } }</pre>	<pre>for (j = (1); j <= *n; ++j) { i__2 = *m; for (i = (1); i <= *m; ++i) { c[(i - (1)) + ((j - (1)) * *ldc)] = (0.); } }</pre>
(a)	(b)

Figure 1: (a) Code snippet from `dgemm`, (b) Unparsed snippet produced by ROSE

Table 1: Output generated by PBound for the code snippet in Figure ?? showing the bytes of data loaded and stored and operations performed

Metric	Computed Estimate
Integer Loads	$4 * (*n + 1 - (1))$
Integer Stores	$4 * (*n + 1 - (1))$
Float Point Loads	0
Float Point Stores	$8 * (*m + 1 - (1)) * (*n + 1 - (1))$
Integer Operations	$(7 * (*m + 1 - (1)) + (3 * (*m + 1 - (1)) + 1) + 1)$
	$(*n + 1 - (1)) + (3 * (*n + 1 - (1)) + 1)$
Floating Point Operations	$(*m + 1 - (1)) * (*n + 1 - (1))$

The graph shows the comparison of the efficiency for the loops in dgemm.c and svmm.c with respect to PBound and the efficiency w.r.t.peak performance. We see that.

3 Future Work and Concluding Remarks

We have shown that PBound can be used to effectively estimate a realistic upper bound on performance of scientific applications. We used it to accurately estimate the implementation efficiency of two scientific codes. PBound currently handles C input and will be extended to handle C++ and ForTran code as well. PBound will also be integrated with OpenAnalysis to perform alias analysis, activity analysis and data flow analysis. Such analysis allows pbound to better estimate the portions of code that will actually be executed. PBounds will also be integrated with a cache model that allows it to estimate data that will be loaded and stored together.