Homework 3

Josh Bradt

February 8, 2016

The timing results for each transpose method are shown in Table 1. These results are also plotted in Figure 1.

In all cases, the cache oblivious code produced the fastest transpose times, though it was still slower than the prediction from the model by a factor of around 4 to 8. For small matrices, the basic transpose outperformed the blocked transpose, but for large matrices, the blocked transpose was faster. This is likely an effect of the extra overhead of the blocked transpose.

The model I used for the prediction was

$$\text{time} = CN^2 \tag{1}$$

where C is the copy time from STREAM, and N is the size of the matrix. The copy time I had gotten from STREAM was 3.141 ms for 3 000 000 elements, so I used a value of $C = (3.141\,\text{ms})/(3\,000\,000\,\text{elements}) = 1.047\,\text{ns/element}$.

Size	Basic [s]	Model [s]	Blocked [s]	Cache Oblivious [s]
20	9.537×10^{-7}	4.188×10^{-7}	1.907×10^{-6}	1×10^{-6}
100	2.790×10^{-5}	1.047×10^{-5}	4.101×10^{-5}	2.9×10^{-5}
500	1.059×10^{-3}	2.618×10^{-4}	1.037×10^{-3}	8.49×10^{-4}
1000	8.120×10^{-3}	1.047×10^{-3}	4.880×10^{-3}	4.93×10^{-3}
1500	1.828×10^{-2}	2.356×10^{-3}	1.147×10^{-2}	1.0964×10^{-2}
2000	3.969×10^{-2}	4.188×10^{-3}	2.061×10^{-2}	2.0465×10^{-2}
4000	1.978×10^{-1}	1.675×10^{-2}	8.718×10^{-2}	8.699×10^{-2}

Table 1: Transpose timing results. The basic and blocked codes were written to run 11 iterations for each array size. The first of these iterations was thrown out to avoid cold start effects, and then the minimum time from the remaining 10 iterations was recorded in this table.

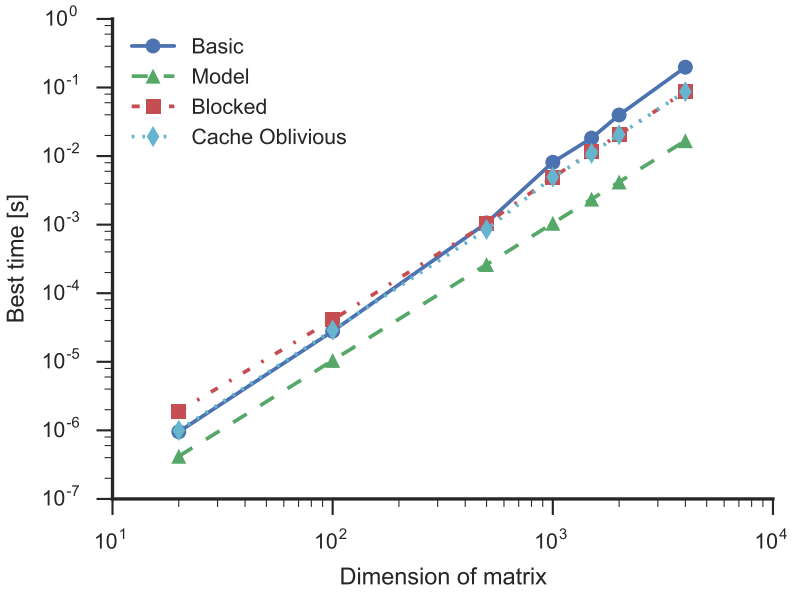

Figure 1: A plot of the results on a log-log scale.