

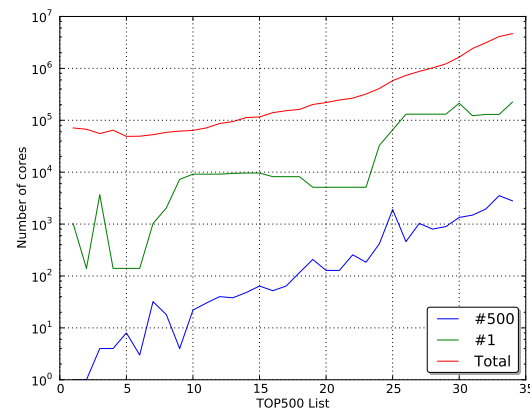
1 Motivation

1.1 Increasing Execution Time

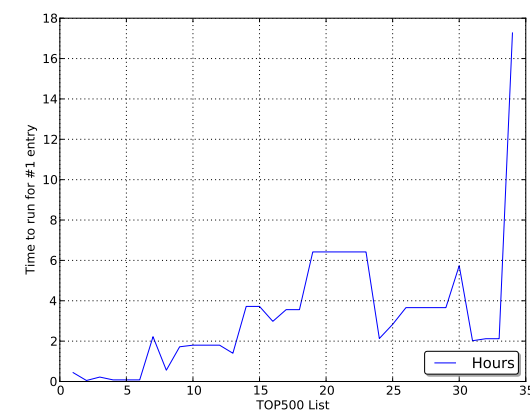
Time to run HPL (t_{Total}) increases because (assuming constant memory per core) the total system memory (M_{Total}) increases:

$$t_{\text{Total}} \propto n^3 = \sqrt{M_{\text{Total}}^3} = N_{\text{cores}}^{3/2} \quad (1)$$

And the number of cores on TOP500 has been increasing at an exponential rate:



Granted the above assumptions, the number of cores on TOP500 may be correlated with the time it takes to run HPL:

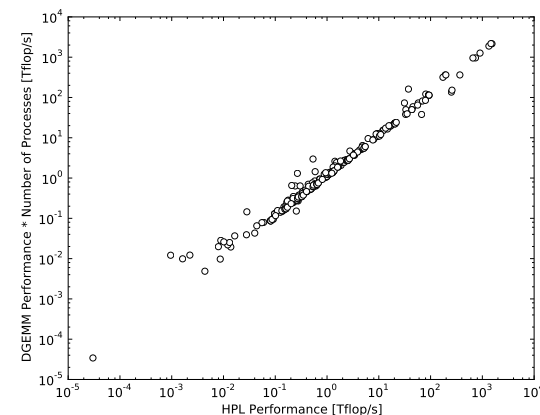


Using hybrid systems with hardware accelerators will not slow this trend because:

- the accelerators use the host memory so the time increases that are related to larger memory size affect both the accelerator and the CPU.
- the accelerators include their own memory that increases with new hardware generations (see, for example, 64-bit memory interface on the Fermi GPUs).

1.2 Insufficient Quality of Existing Models

A common way to predict execution time of full scale experiments is through performance modeling. Very simple performance models cannot accurately predict the time to run HPL. For example, even though it's well correlated, single-process matrix-matrix multiply routine (DGEMM) is not accurate enough of a predictor: median prediction error is 15%. Here is the correlation graph based on data from the HPC Challenge's public database:



Performance models based on memory access patterns alone do not capture the computational intensity of HPL – a crucial feature of HPL.

Performance models based on HPL's individual kernels are

- complicated: they require exact and adequate measurement of performance of component kernels, and they
- introduce unpredictable error due non-linear and overlapping effects that occur when the performance of individual components is combined.

2 Goals

In addition to solving the problems mentioned above (reducing the time to run HPL and allow predictions of execution time), the following goals have to be satisfied:

Simplicity easy to explain and implement.

Stress-testing the time to run should still present a significant challenge for the machine (it has to last, say, 12 hours).

Verification The results have to be numerically verifiable.

Full system utilization Even if doing a partial run the full matrix has to be used.

Interpolation The rate of execution from the shorten run can never exceed the rate from a complete run.

3 Sampled Execution

First, for simplicity, consider LU factorization without any pivoting

$$A = LU. \quad (2)$$

We'd like to perform *partial execution*: only some parts of the matrix will be factored. For this, let's assume the following structure of A:

$$A = \begin{bmatrix} I & A_{12} & 0 & A_{14} & 0 \\ 0 & A_{22} & 0 & A_{24} & 0 \\ 0 & A_{32} & I & A_{34} & 0 \\ 0 & A_{42} & 0 & A_{44} & 0 \\ 0 & A_{52} & 0 & A_{54} & I \end{bmatrix}. \quad (3)$$

The result of a sampled factorization should be as follows:

$$A = \begin{bmatrix} I & A_{12} & \tilde{A}_{13} & A_{14} & \tilde{A}_{15} \\ 0 & D_{22} & \tilde{A}_{23} & U_{24} & \tilde{A}_{25} \\ 0 & L_{32} & \tilde{A}_{33} & U_{34} & \tilde{A}_{35} \\ 0 & L_{42} & \tilde{A}_{43} & D_{44} & \tilde{A}_{45} \\ 0 & L_{52} & \tilde{A}_{53} & L_{54} & \tilde{A}_{55} \end{bmatrix}. \quad (4)$$

The updates can be numerically verified with the following system:

$$\begin{bmatrix} I & A_{12} & A_{13} & A_{14} & A_{15} \\ 0 & A_{22} & 0 & A_{24} & 0 \\ 0 & A_{32} & I & A_{34} & 0 \\ 0 & A_{42} & 0 & A_{44} & 0 \\ 0 & A_{52} & 0 & A_{54} & I \end{bmatrix} X = \begin{bmatrix} 0 & 0 \\ A_{23} & A_{25} \\ A_{33} & A_{35} \\ A_{43} & A_{45} \\ A_{53} & A_{55} \end{bmatrix}$$

For numerical stability, HPL performs LU factorization with partial pivoting:

$$PA = LU, \quad (5)$$

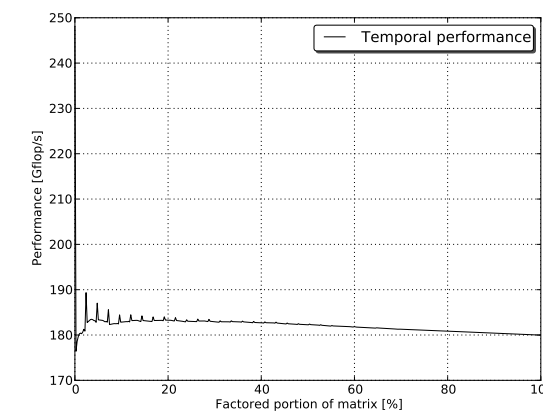
Introduction of the pivoting matrix has to be reconciled with partial execution. The

partitioning of the permutation matrix becomes:

$$P = \begin{bmatrix} I & 0 & 0 & 0 & 0 \\ 0 & P_{22} & P_{23} & P_{24} & P_{25} \\ 0 & P_{32} & I & 0 & 0 \\ 0 & P_{42} & 0 & P_{44} & P_{45} \\ 0 & P_{52} & 0 & P_{54} & I \end{bmatrix}. \quad (6)$$

4 Choosing the Portion of HPL to Execute

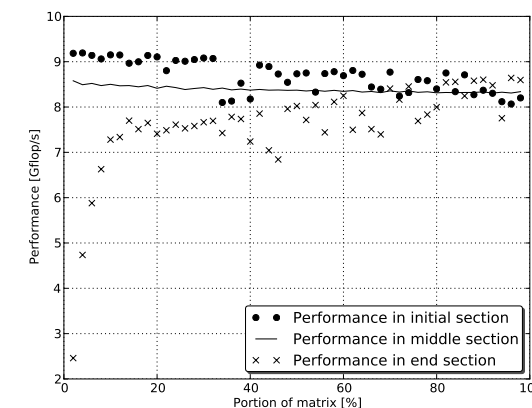
Direct dense linear algebra codes such as HPL commonly exhibit non-linear variation in temporal performance:



Three simple choices of portions of HPL to execute are the initial, the middle, and the end section of the system matrix A:

$$A = [A_{\text{initial}} \ 0 \ A_{\text{middle}} \ 0 \ A_{\text{end}}]. \quad (7)$$

The achieved performance of sampled factorization varies with the relative size of the portion of the matrix used for factorization:



Only the end section guarantees that the performance will not be overestimated.

5 Performance Model

The performance reported by HPL is the ratio of the operation count and the time to perform the solve:

$$R(n) = \frac{\text{Op}_{\text{count}}}{t_{\text{Total}}} = \frac{2/3n^3 + 3/2n^2}{t_{\text{Total}}(n)} \quad (8)$$

The operation count is fixed regardless of the underlying algorithm to facilitate performance comparisons. The time to run HPL can be modeled as:

$$t(n) = an^3 + bn^2 + cn + d \quad (9)$$

where a , b , c , and d can be obtained experimentally.

Given a set of 4 experiments with varying problem sizes n_1 , n_2 , n_3 , and n_4 we obtain the actual running times t_1 , t_2 , t_3 , and t_4 . Using the results of these experiments, we formulate the problem as a system of linear equations:

$$\begin{bmatrix} n_1^3 & n_1^2 & n_1 & 1 \\ n_2^3 & n_2^2 & n_2 & 1 \\ \dots & \dots & \dots & \dots \\ n_k^3 & n_k^2 & n_k & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \\ \dots \\ t_k \end{bmatrix} \quad (10)$$

We use least-squares method for obtaining a solution: the coefficients for our model of HPL's execution time.

6 Performance Results

Combination of partial execution and our performance model allows for accurate prediction of execution time (modeling error of 1% or less) on tens of thousands of cores. And allows for adjustment of execution time according to the desired performance goal:

