Subject: Speed does matter
Posted by on Wed, 03 Oct 2012 08:17:49 GMT
View Forum Message <> Reply to Message

It started while using the same algorithm coded similarly (we hope) in Matlab and IDL, heavy with linear algebra and with very larges matrix. Unfortunately we got huge performances discrepancies, I mean not in the two or three multiple but sometimes in the fifty range !

Then we tried to find some existing benchmarks and found a very partial one here (http://fwenvi-idl.blogspot.com/2011/10/numpy-is-fast.html) for multiplying large matrix, in which IDL outperforms Python by a factor of 27.
But this is not the end of the story, we learned then in the same blog article of the existence of Numpy compiled against the Intel Math Kernel Library. With these binaries now Python outperforms IDL by a factor greater than 10 !

These incredible numbers prompted us to pit this MKL flavor of Python/Numpy 32 bits against IDL 64 bits, here are some results obtained on a recent Windows 7 computer with 4 cores, unfortunately without the original plots.

 *** ********************
Multiplication of FLOAT32 Matrix (*) : IDL speed multiplicative of ×1.5
Multiplication of FLOAT64 Matrix (*) : IDL speed multiplicative of ×1.4
Multiplication of Complex Matrix (*) : IDL speed multiplicative of ×2.8

Matrix multiplication of FLOAT32 Matrix (#) : Python/Numpy MKL speed multiplicative of ×12
Matrix multiplication of FLOAT64 Matrix (#) : Python/Numpy MKL speed multiplicative of ×6
Matrix multiplication of Complex Matrix (#) : Python/Numpy MKL speed multiplicative of ×3.1

FFT of FLOAT32 Matrix : IDL speed multiplicative of ×1.2
FFT of FLOAT64 Matrix : roughly equivalent
FFT of Complex Matrix : Python/Numpy MKL speed multiplicative of ×1.6

The inverse of a square array (with LAPACK routines in both cases) case is harder to present with a synthetic number since the slope is different.
Invert of FLOAT32 Matrix : Python/Numpy MKL speed multiplicative of ×1.4 at 100×100 size increasing to ×20 at 1200×1200 size.
Invert of FLOAT64 Matrix : Python/Numpy MKL speed multiplicative of ×3.0 at 100×100 size increasing to ×28 at 700×700 size.
Invert of Complex Matrix : Python/Numpy MKL speed multiplicative of ×2.7 at 100×100 size increasing to ×9 at 800×800 size.

The Singular Value Decomposition of a square array (with LAPACK routines in both cases) case is also harder to present with a synthetic number since the slope is different.
SVD of FLOAT32 Matrix : Python/Numpy MKL speed multiplicative of ×1.4 at 100×100 size increasing to ×20 at 1200×1200 size
SVD of FLOAT64 Matrix : Python/Numpy MKL speed multiplicative of ×0.73 at 100×100 size increasing to ×28 at 1800×1800 size.
SVD of Complex Matrix : Python/Numpy MKL speed multiplicative of ×3.8 at 100×100 size

increasing to ×18 at 1500×1500 size.

*** ********************

How is it achieved ? Well, you could look at your CPU performance tab while running Python MKL, all cores are at 100 %, while most IDL routines hardly top 30 % on a 4-core computer. This can explain part of the performance gain.

Similar benchmarks were computed on another computer between Python/Numpy MKLPython/Numpy MKL and Matlab, demonstrating other artifacts but mostly with "comparable" performances (in particular with Python MKL 64 bits). These results highlight the incredible performances impact of the Intel Math Kernel Library, in particular here in linear algebra routines. Since this Library is a Royalty-free, per developer licensing, I'd dream to see a future IDL compilation against such Library.
Any chances ?