Distributed Computing and Introduction to High Performance Computing

Ahmed Ratnani[1]
[1]Mohammed VI Polytechnic University, Benguerir, Morocco

Outline of this lecture

- Matrix multiplication example
- Profiling a Python code
- Accelerate a Python code
  - Using Numpy
  - Using Cython
  - Using Numba
  - Using Pyccel

Matrix multiplication example

See the following Jupyter Notebook on github:
Advanced-Python-Motivations

Profiling a Python code

Why?

- It's mandatory to know what sections of code are bottlenecks in order to improve performance.
- You need to measure it, not to guess it
- **Premature optimization is the root of all evil** D. Knuth
- **First make it work. Then make it right. Then make it fast**. K. Beck
- How?

Profiling a Python code

Different kinds of profilers

- Deterministic and statistical profiling
  - the profiler will be monitoring all the events
  - it will sample after time intervals to collect that information
- The level at which resources are measured; module, function or line level
- Profile visualizers

Profiling a Python code

Available tools

- **Inbuilt timing modules**
- **profile and cProfile**
- pstats
- **line_profiler**
- Yappi
- vmprof-python
- pyinstrument
- gprof2dot
- pyprof2calltree, KCacheGrind
- **snakeviz**
- Scalene

Profiling a Python code

Use case

```python
def linspace(start, stop, n):
  step = float(stop - start) / (n - 1)
  return [start + i * step for i in range(n)]

def mandel(c, maxiter):
  z = c
  for n in range(maxiter):
    if abs(z) ¿ 2:
      return n
    z = z*z + c
  return n

def mandel_set(xmin=-2.0, xmax=0.5, ymin=-1.25, ymax=1.25,
                width=1000, height=1000, maxiter=80):
  r = linspace(xmin, xmax, width)
  i = linspace(ymin, ymax, height)
  n = [[0]*width for _ in range(height)]
  for x in range(width):
    for y in range(height):
      n[y][x] = mandel(complex(r[x], i[y]), maxiter)
  return n
```

Profiling a Python code

timeit

The very naive way

```
import time

start_time = time.time()
mandel_set()
end_time = time.time()
# Time taken in seconds
elapsed_time = end_time - start_time

print('> Elapsed time', elapsed_time)
```

or using the magic method `timeit`

```
  [In] %timeit mandel_set()
  [Out] 3.01 s +/- 84.6 ms per loop (mean +/- std. dev. of 7 runs, 1 loop each)
```

Profiling a Python code

prun

```
[In] %prun -s cumulative mandel_set()
```

which is, in console mode, equivalent to

```
python -m cProfile -s cumulative mandel.py
```

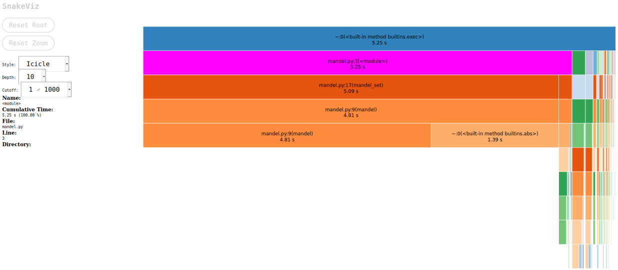
```
          25214601 function calls in 5.151 seconds

   Ordered by: cumulative time

   ncalls tottime percall cumtime percall filename:lineno(function)
        1 0.000 0.000 5.151 5.151 {built-in method builtins.exec}
        1 0.002 0.002 5.151 5.151 <string>:1(<module>)
        1 0.291 0.291 5.149 5.149 <ipython-input-4-9421bc2016cb>:13(mandel_set)
  1000000 3.461 0.000 4.849 0.000 <ipython-input-4-9421bc2016cb>:5(mandel)
 24214592 1.388 0.000 1.388 0.000 {built-in method builtins.abs}
        1 0.008 0.008 0.008 0.008 <ipython-input-4-9421bc2016cb>:17(<listcomp>)
        2 0.000 0.000 0.000 0.000 <ipython-input-4-9421bc2016cb>:1(linspace)
        2 0.000 0.000 0.000 0.000 <ipython-input-4-9421bc2016cb>:3(<listcomp>)
        1 0.000 0.000 0.000 0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

- Most of the time is spent in the `mandel` function
- profiling introduces some overhead 5.14(s) instead of 3.01(s)

Visualization

- Profiling results can be visualized with SnakeViz
- We must be in console mode

```
python3 -m cProfile -o mandel.prof mandel.py
snakeviz --port 6542 --hostname localhost --server mandel.prof
```



ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1000000	3.415	3.415e-06	4.807	4.807e-06	mandel.py:9(mandel)
24214688	1.393	5.752e-08	1.393	5.752e-08	~:0(<built-in method builtins.abs>)
1	0.2763	0.2763	5.087	5.087	mandel.py:17(mandel_set)
316	0.03232	0.0001023	0.03232	0.0001023	~:0(<built-in method builtins.compile>)

Profiling a Python code

Details at the line level

- We know that most of the time is spent in the `mandel` function
- We shall use the `line_profiler` package on this function to get details at the line level

```
[In] %load_ext line_profiler
[In] %lprun -f mandel mandel_set()
```

Which gives the result

```
Timer unit: 1e-06 s

Total time: 22.8401 s
File: <ipython-input-2-9421bc2016cb>
Function: mandel at line 5

#Line       Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     5                                           def mandel(c, maxiter):
     6    1000000     250304.0      0.3      1.1      z = c
     7   24463110    6337732.0      0.3     27.7      for n in range(maxiter):
     8   24214592    8327289.0      0.3     36.5          if abs(z) > 2:
     9     751482     201108.0      0.3      0.9              return n
    10   23463110    7658255.0      0.3     33.5          z = z*z + c
```

Profiling a Python code

Details at the line level

This can be done in console mode as well

```
import line_profiler

@profile
def mandel(c, maxiter):
  z = c
  for n in range(maxiter):
    if abs(z) ¿ 2:
      return n
    z = z*z + c
  return n
```

Then on the command line

```
kernprof -l -v mandel.py
```

Accelerating a Python code

Numpy

Default BLAS - LAPACK

```
Dotted two 4096x4096 matrices in 64.22 s.
Dotted two vectors of length 524288 in 0.80 ms.
SVD of a 2048x1024 matrix in 10.31 s.
Cholesky decomposition of a 2048x2048 matrix in 6.74 s.
Eigendecomposition of a 2048x2048 matrix in 53.77 s.
```

ATLAS

```
Dotted two 4096x4096 matrices in 3.46 s.
Dotted two vectors of length 524288 in 0.73 ms.
SVD of a 2048x1024 matrix in 2.02 s.
Cholesky decomposition of a 2048x2048 matrix in 0.51 s.
Eigendecomposition of a 2048x2048 matrix in 29.90 s.
```

Accelerating a Python code

Numpy

Intel MKL

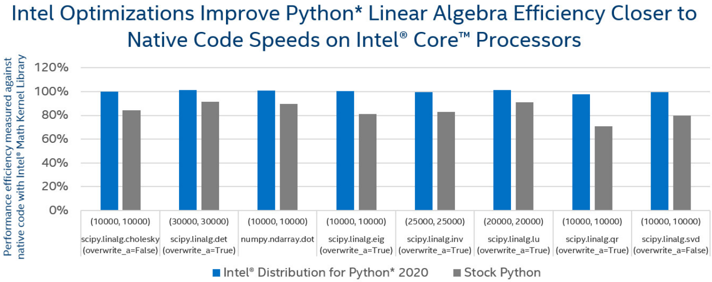
```
Dotted two 4096x4096 matrices in 2.44 s.
Dotted two vectors of length 524288 in 0.75 ms.
SVD of a 2048x1024 matrix in 1.34 s.
Cholesky decomposition of a 2048x2048 matrix in 0.40 s.
Eigendecomposition of a 2048x2048 matrix in 10.07 s.
```

OpenBLAS

```
Dotted two 4096x4096 matrices in 3.97 s.
Dotted two vectors of length 524288 in 0.74 ms.
SVD of a 2048x1024 matrix in 1.96 s.
Cholesky decomposition of a 2048x2048 matrix in 0.46 s.
Eigendecomposition of a 2048x2048 matrix in 32.95 s.
```

Accelerating a Python code

Numpy using MKL



Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer. Performance results are based on testing as of **November 27, 2019** and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

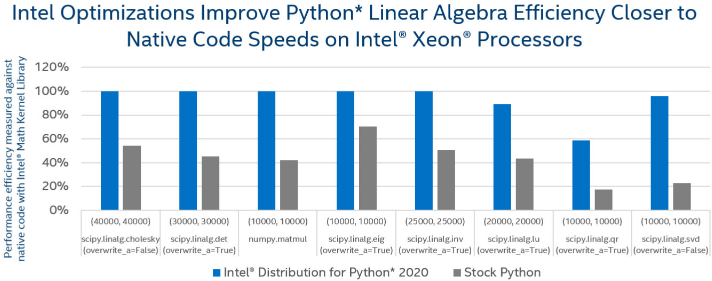
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Configuration: Testing by Intel as of **November 27, 2019**. Stock Python: python 3.7.5 h0371630_0 installed from conda, numpy 1.17.4, numba 0.46.0, llvmlite 0.30.0, scipy 1.3.2, scikit-learn 0.21.3 installed from pip; Intel Python: Intel® Distribution for Python* 2020 Gold: python 3.7.4 hf484d3e_3, numpy 1.17.3 py37ha68da19_4, mkl 2020 intel_133, mkl_fft 1.0.15 py37ha68da19_3, mkl_random 1.1.0 py37ha68da19_0, numba 0.45.1 np117py37_1, llvmlite 0.29.0 py37hf484d3e_9, scipy 1.3.1 py37ha68da19_2, scikit-learn 0.21.3 py37ha68da19_14, daal 2020 intel_133, daal4py 2020 py37ha68da19_4; CentOS Linux 7.4.1708, kernel 3.10.0-693.el7.x86_64; Hardware: Intel(R) Core(TM) i7-7567U CPU @ 3.50GHz (1 socket, 2 cores/socket, HT:2), 32 GB of DDR4 RAM, 2 DIMMs of 16 GB@2133MHz

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Accelerating a Python code

Numpy using MKL



Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer. Performance results are based on testing as of **November 27, 2019** and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

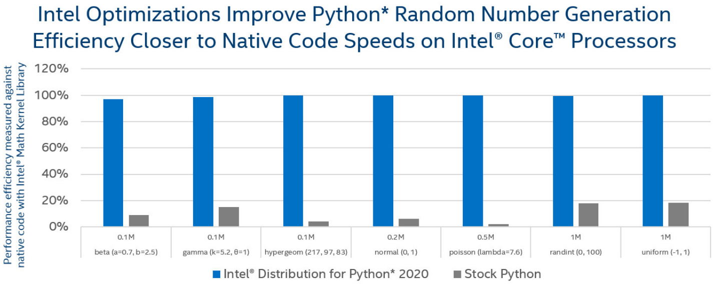
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Configuration: Testing by Intel as of **November 27, 2019**. Stock Python: python 3.7.5 h0371630_0 installed from conda, numpy 1.17.4, numba 0.46.0, llvmlite 0.30.0, scipy 1.3.2, scikit-learn 0.21.3 installed from pip; Intel Python: Intel® Distribution for Python* 2020 Gold: python 3.7.4 hf484d3e_3, numpy 1.17.3 py37ha68da19_4, mkl 2020 intel_133, mkl_fft 1.0.15 py37ha68da19_3, mkl_random 1.1.0 py37ha68da19_0, numba 0.45.1 np117py37_1, llvmlite 0.29.0 py37hf484d3e_9, scipy 1.3.1 py37ha68da19_2, scikit-learn 0.21.3 py37ha68da19_14, daal 2020 intel_133, daal4py 2020 py37ha68da19_4; CentOS Linux 7.3.1611, kernel 3.10.0-514.el7.x86_64; Hardware: Intel(R) Xeon(R) Platinum 8280 CPU @ 2.70GHz (2 sockets, 28 cores/socket, HT: off), 256 GB of DDR4 RAM, 16 DIMMs of 16 GB@2666MHz

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Accelerating a Python code

Numpy using MKL



Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer. Performance results are based on testing as of **November 27, 2019** and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

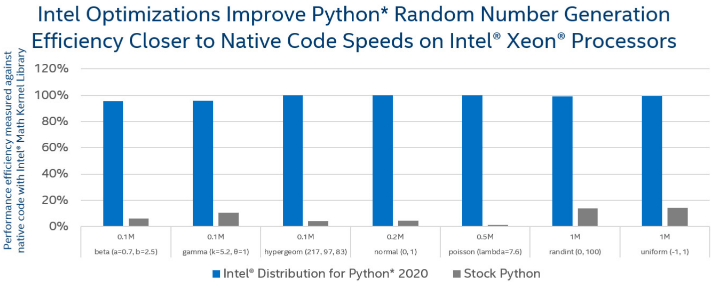
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Configuration: Testing by Intel as of **November 27, 2019**. Stock Python: python 3.7.5 h0371630_0 installed from conda, numpy 1.17.4, numba 0.46.0, llvmlite 0.30.0, scipy 1.3.2, scikit-learn 0.21.3 installed from pip; Intel Python: Intel® Distribution for Python* 2020 Gold: python 3.7.4 hf484d3e_3, numpy 1.17.3 py37ha68da19_4, mkl 2020 intel_133, mkl_fft 1.0.15 py37ha68da19_3, mkl_random 1.1.0 py37ha68da19_0, numba 0.45.1 np117py37_1, llvmlite 0.29.0 py37hf484d3e_9, scipy 1.3.1 py37ha68da19_2, scikit-learn 0.21.3 py37ha68da19_14, daal 2020 intel_133, daal4py 2020 py37ha68da19_4; CentOS Linux 7.4.1708, kernel 3.10.0-693.el7.x86_64; Hardware: Intel(R) Core(TM) i7-7567U CPU @ 3.50GHz (1 socket, 2 cores/socket, HT:2), 32 GB of DDR4 RAM, 2 DIMMs of 16 GB@2133MHz

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Accelerating a Python code

Numpy using MKL



Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer. Performance results are based on testing as of **November 27, 2019** and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

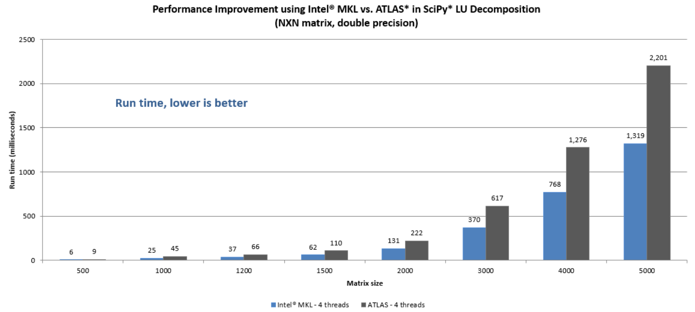
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Configuration: Testing by Intel as of **November 27, 2019**. Stock Python: python 3.7.5 h0371630_0 installed from conda, numpy 1.17.4, numba 0.46.0, llvmlite 0.30.0, scipy 1.3.2, scikit-learn 0.21.3 installed from pip; Intel Python: Intel® Distribution for Python* 2020 Gold: python 3.7.4 hf484d3e_3, numpy 1.17.3 py37ha68da19_4, mkl 2020 intel_133, mkl_fft 1.0.15 py37ha68da19_3, mkl_random 1.1.0 py37ha68da19_0, numba 0.45.1 np117py37_1, llvmlite 0.29.0 py37hf484d3e_9, scipy 1.3.1 py37ha68da19_2, scikit-learn 0.21.3 py37ha68da19_14, daal 2020 intel_133, daal4py 2020 py37ha68da19_4; CentOS Linux 7.3.1611, kernel 3.10.0-514.el7.x86_64; Hardware: Intel(R) Xeon(R) Platinum 8280 CPU @ 2.70GHz (2 sockets, 28 cores/socket, HT: off), 256 GB of DDR4 RAM, 16 DIMMs of 16 GB@2666MHz

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Accelerating a Python code

Numpy using MKL



Configuration Info - Versions: Intel® Math Kernel Library (Intel® MKL) 11.1 update 1, ATLAS* 3.10.1 with LAPACK 3.4.2, NumPy* 1.8.0, SciPy* 0.13.2, Python 3.3; Hardware: Intel® Core® i5-4670T Processor (6 MB LLC, 2.30Ghz), 4 GB of RAM; Operating System: Fedora 16 x86_64; Benchmark Source: Intel Corporation.

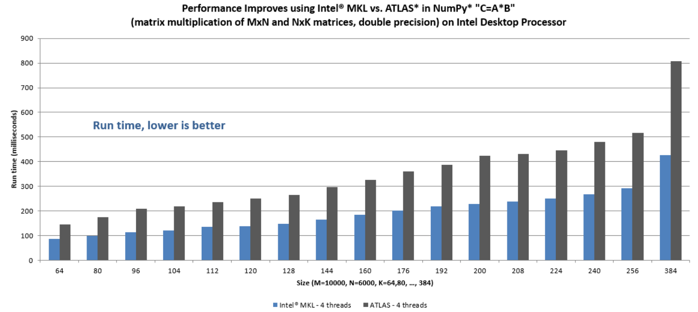
Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, refer to www.intel.com/performance/resources/benchmark_limitations.htm.

* Other brands and names are the property of their respective owners

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Accelerating a Python code

Numpy using MKL



Configuration Info - Versions: Intel® Math Kernel Library (Intel® MKL) 11.1 update 1, ATLAS* 3.10.1 with LAPACK 3.4.2, NumPy* 1.8.0, SciPy* 0.13.2, Python 3.3; Hardware: Intel® Core® i5-4670T Processor (6 MB LLC, 2.30Ghz), 4 GB of RAM; Operating System: Fedora 16 x86_64; Benchmark Source: Intel Corporation.

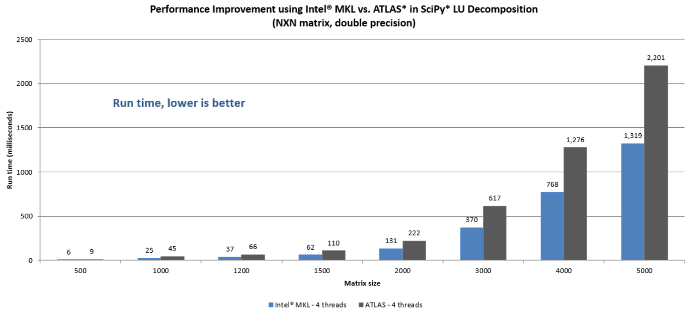
Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, refer to www.intel.com/performance/resources/benchmark_limitations.htm.

* Other brands and names are the property of their respective owners

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Accelerating a Python code

Numpy using MKL



Configuration Info - Versions: Intel® Math Kernel Library (Intel® MKL) 11.1 update 1, ATLAS* 3.10.1 with LAPACK 3.4.2, NumPy* 1.8.0, SciPy* 0.13.2, Python 3.3; Hardware: Intel® Core® i5-4670T Processor (6 MB LLC, 2.30Ghz), 4 GB of RAM; Operating System: Fedora 16 x86_64; Benchmark Source: Intel Corporation.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, refer to www.intel.com/performance/resources/benchmark_limitations.htm.

* Other brands and names are the property of their respective owners

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804