PUMPING PYTHON PERFORMANCE

Sergey Maidanov

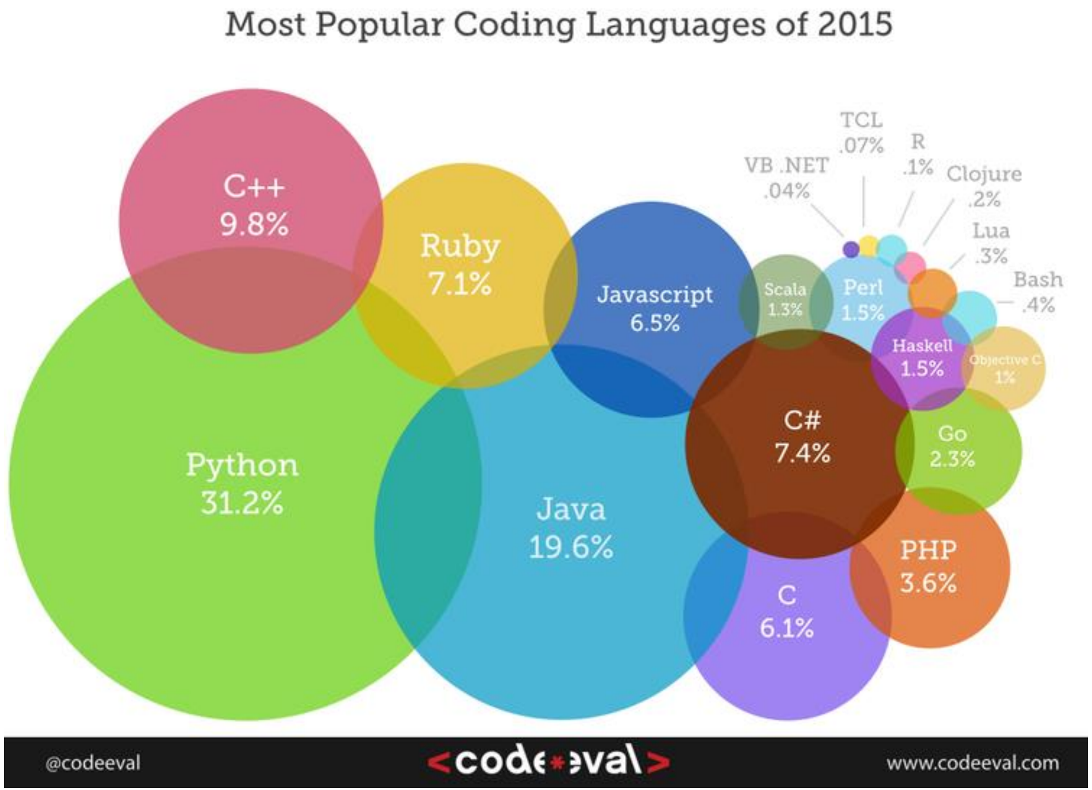
Software Engineering Manager,

Scripting Analyzers & Tools, Intel

Programming Languages by Popularity

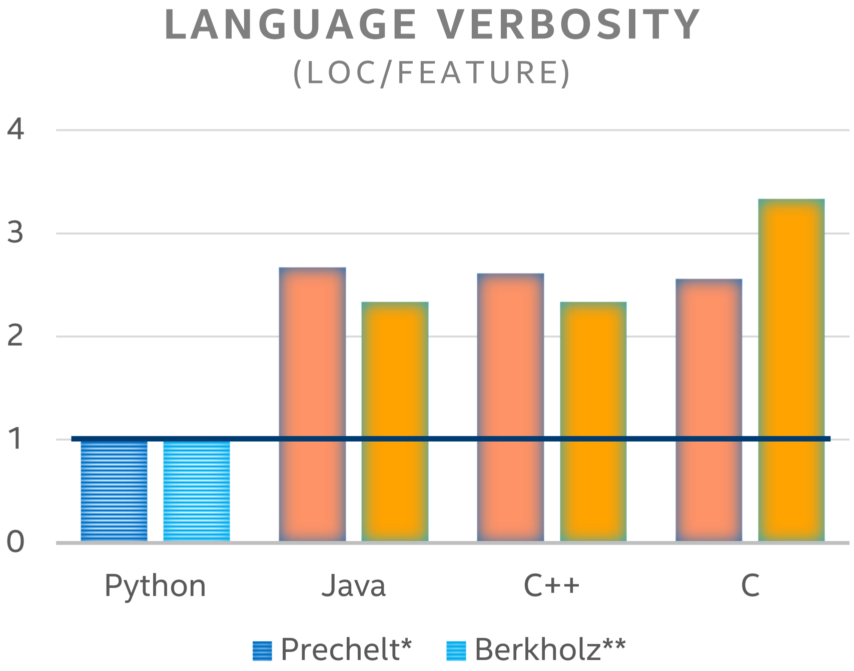
Python remains **#1** programming language in **hiring demand** followed by **Java** and **C++**

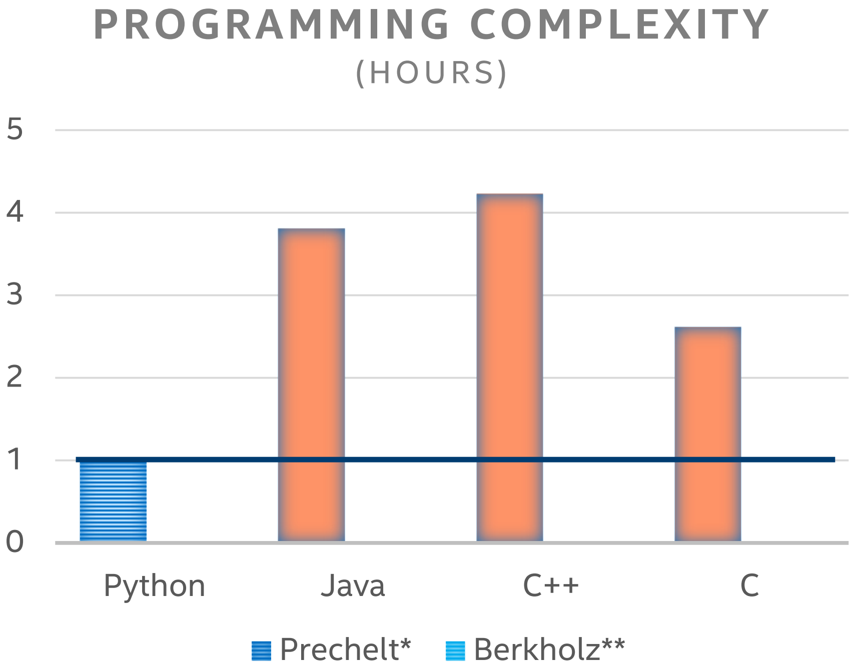
Go and **Scala** demonstrate **strong growth** for last 2 years



* Source: CodeEval, Feb 2015

Programming Languages Productivity





* L.Prechelt, An empirical comparison of seven programming languages, IEEE Computer, 2000, Vol. 33, Issue 10, pp. 23-29
** RedMonk - D.Berkholz, Programming languages ranked by expressiveness

Numerical Modeling: From Prototype To Production

Workstation

Prototyping

Development cost

1x

- 1. Pre-Processing
- 2. Analysis
- 3. Modeling
- 4. Model Validation
- 5. Visualization

Python*, R*, Matlab*, Excel*

HPC/Big Data Cluster

Production

Development cost

3-10x and more

- 1. Pre-processing
- 2. Model Calibration
- 3. Decision Making
- 4. Model Validation
- 5. Reporting

Fortran*, C++. Java*/Scala*

MPI*, OpenMP*, Hadoop*, Spark*, …

Why's Interpreted Code Unfriendly To Modern HW?

Moore's law still works and will work for at least next 10 years

We have hit limits in

- Power
- Instruction level parallelism
- Clock speed

But not in

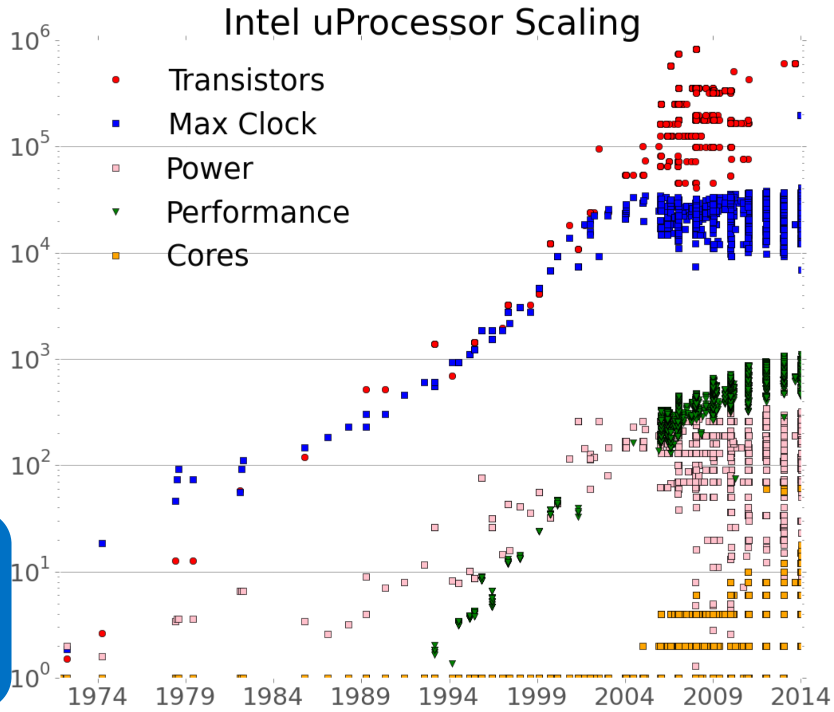
- Transistors (more memory, bigger caches, wider SIMD, specialized HW)
- Number of cores

Flop/Byte continues growing

- 10x worse in last 20 years

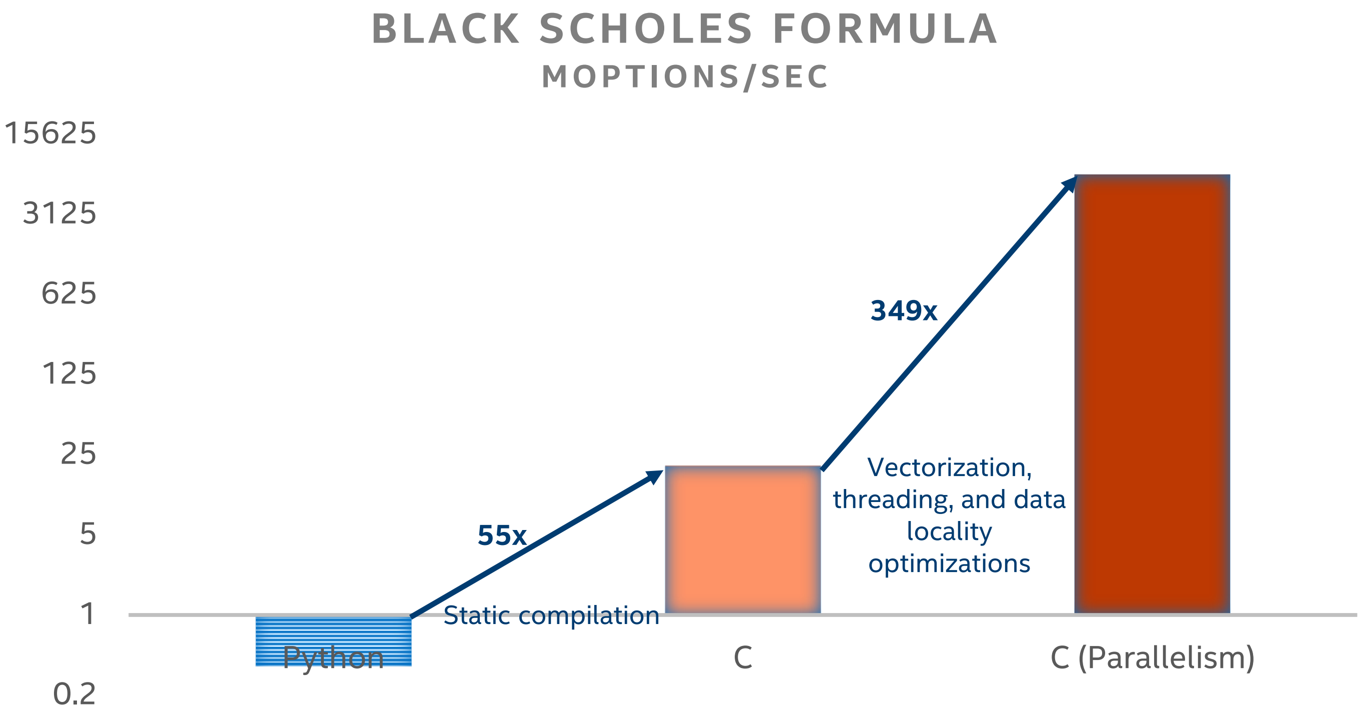
Efficient software development means

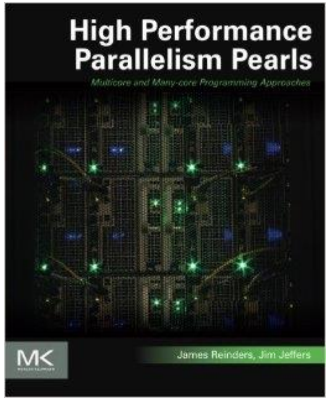
- Optimizations for data locality & contiguity
- Vectorization
- Threading



Source: J.Cownie, HPC Trends and What They Mean for Me, Imperial College, Oct. 2015

Performance: Native vs. Interpreted Code





Chapter 19. Performance Optimization of Black Scholes Pricing

$$V_{call} = S_0 \cdot \mathrm{CDF}(d_1) - e^{-rT} \cdot X \cdot \mathrm{CDF}(d_2)$$

$$V_{put} = e^{-rT} \cdot X \cdot \mathrm{CDF}(-d_2) - S_0 \cdot \mathrm{CDF}(-d_1)$$

$$d_1 = \frac{\ln\left(\frac{S_0}{X}\right) + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}$$

$$d_2 = \frac{\ln\left(\frac{S_0}{X}\right) + \left(r - \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}$$

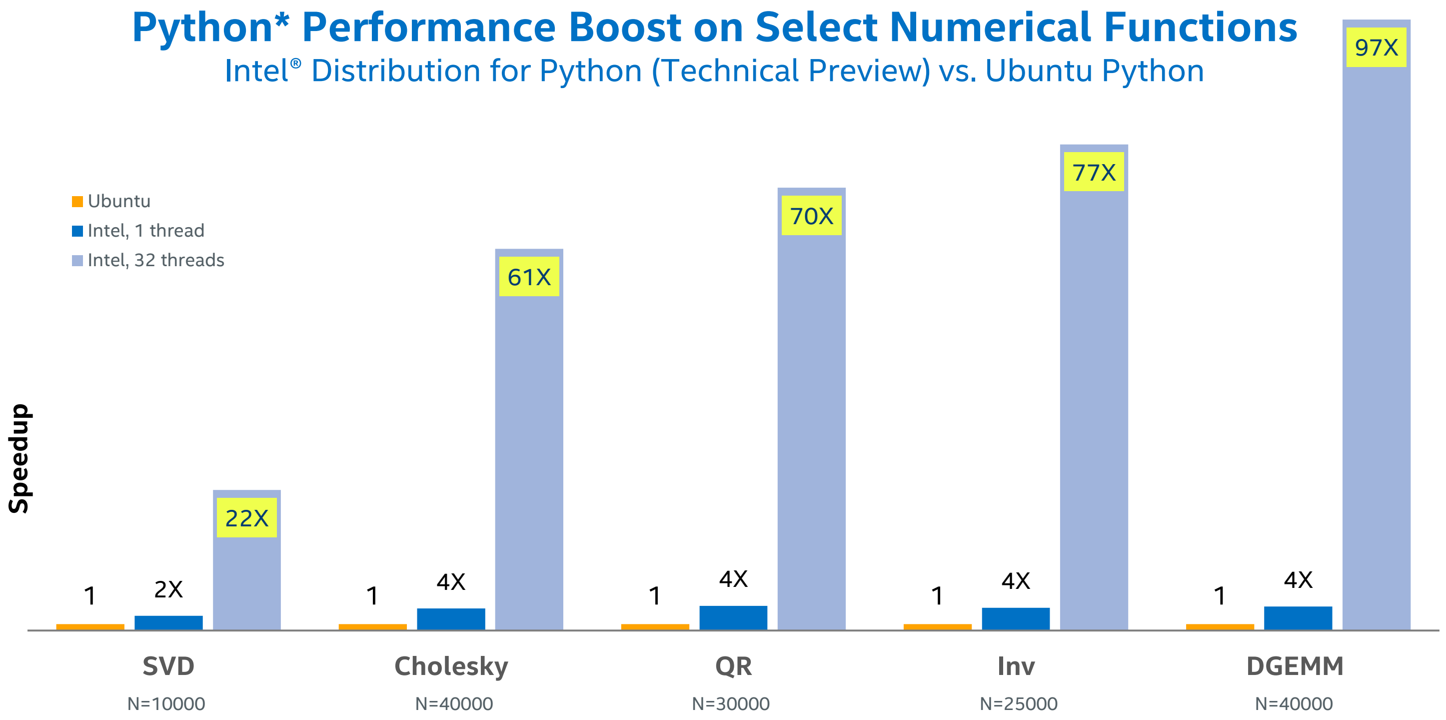
Configuration info: - Versions: Intel® Distribution for Python 2.7.10 Technical Preview 1 (Aug 03, 2015), icc 15.0; Hardware: Intel® Xeon® CPU E5-2698 v3 @ 2.30GHz (2 sockets, 16 cores each, HT=OFF), 64 GB of RAM, 8 DIMMS of 8GB@2133MHz; Operating System: Ubuntu 14.04 LTS.

Possible Improvement Approaches

How to make Python more usable in both prototyping & production?

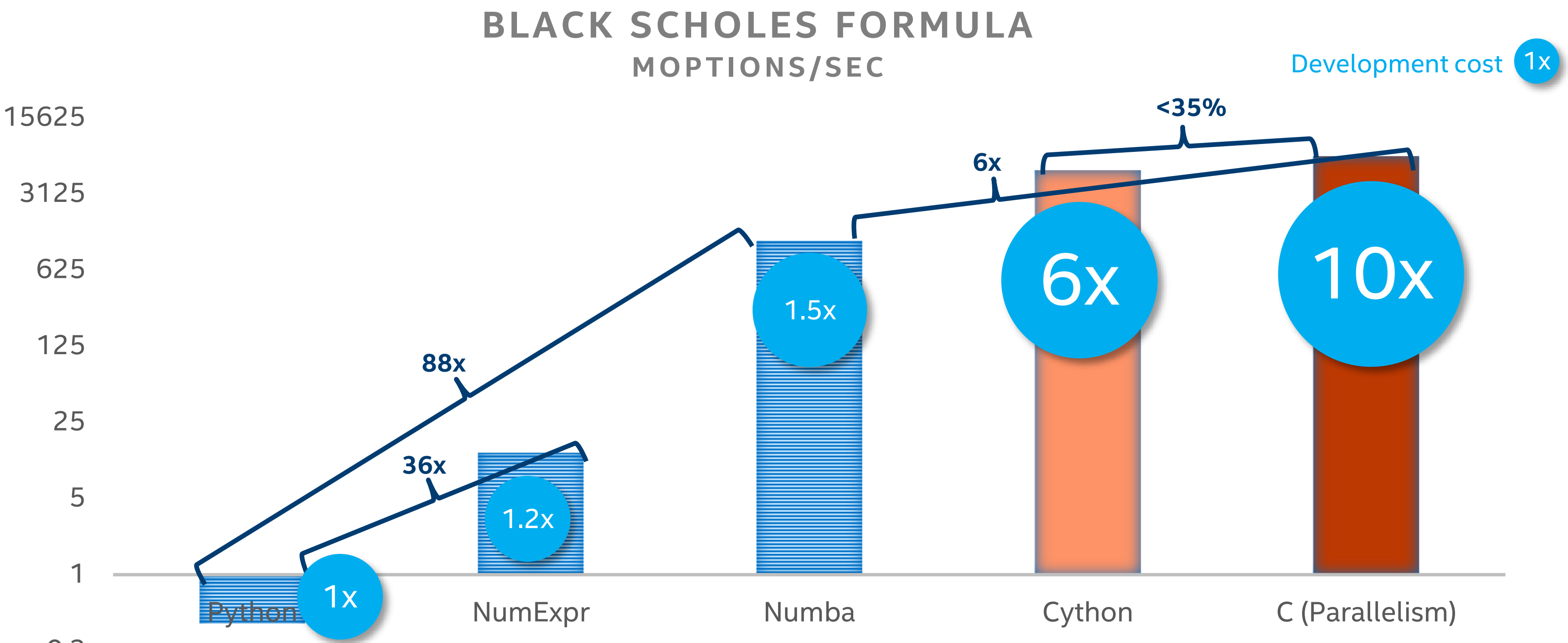
- Numerical/machine learning packages (NumPy/SciPy/Scikit-learn) accelerated with native libraries (e.g. Intel® MKL)
- Python language extensions that exploit vectorization and multi-core parallelism, e.g. Cython (via GCC/ICC), Numba (LLVM)
- Better performance profiling of Python codes
- Packages and extensions for multi-node parallelism, e.g. mpi4py
- Integration with Big Data/ML infrastructures (Hadoop, Spark)

Python Numerical Packages Acceleration



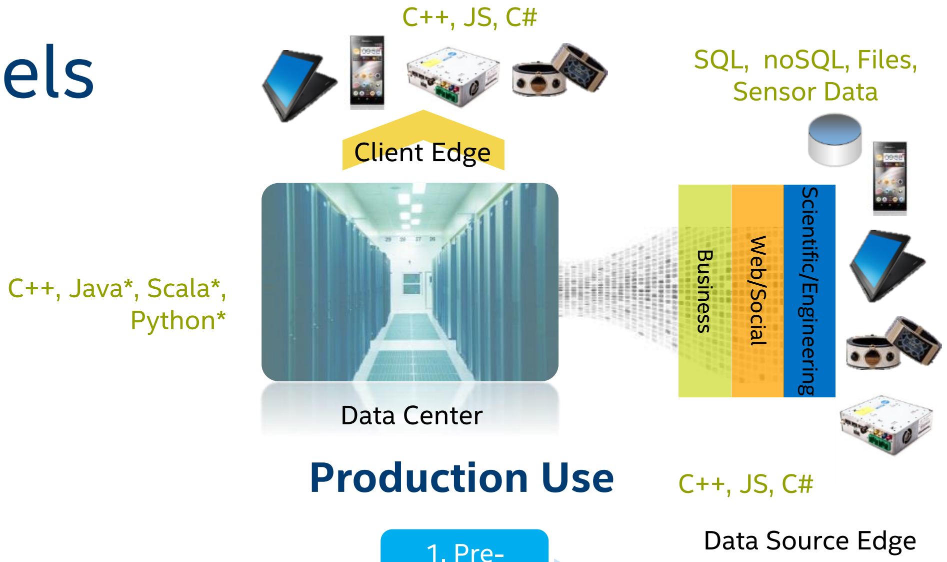
Configuration info: - Versions: Intel® Distribution for Python 2.7.10 Technical Preview 1 (Aug 03, 2015), Ubuntu* built Python*: Python 2.7.10, NumPy 1.9.2 built with gcc 4.8.4; Hardware: Intel® Xeon® CPU E5-2698 v3 @ 2.30GHz (2 sockets, 16 cores each, HT=OFF), 64 GB of RAM, 8 DIMMS of 8GB@2133MHz; Operating System: Ubuntu 14.04 LTS.

Python Language Extensions For Performance



Configuration info: - Versions: Intel® Distribution for Python 2.7.10 Technical Preview 1 (Aug 03, 2015), Ubuntu* built Python*: Python 2.7.10, NumPy 1.9.2 built with gcc 4.8.4, llvm 3.6.0, Numba 0.22.0, icc 15.0; Hardware: Intel® Xeon® CPU E5-2698 v3 @ 2.30GHz (2 sockets, 16 cores each, HT=OFF), 64 GB of RAM, 8 DIMMS of 8GB@2133MHz; Operating System: Ubuntu 14.04 LTS.

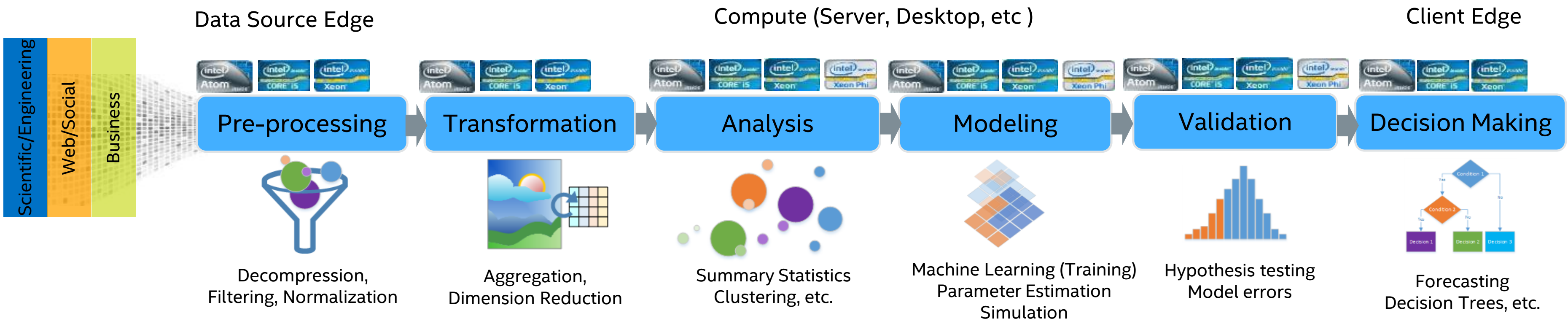
Data Analytics Usage Models



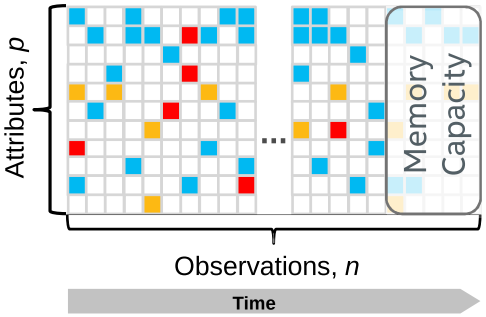
Big Data Requires End-To-End Solutions

What device do I run analytics?

- Perform analysis close to data source to optimize response latency, decrease network bandwidth utilization, and maximize security.
- Offload data to cluster for large-scale analytics only.
- Make personalized decisions on a client device



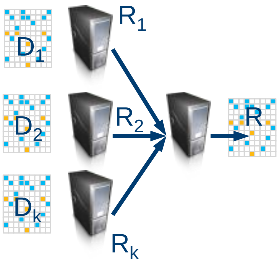
Big Data Flow and Computational Flow



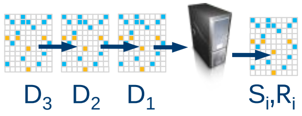
- Numeric
- Categorical
- Blank/Missing
- Outlier

Big Data Attributes	Computational Solution
Distributed across different nodes/devices	•Distributed computing, e.g. comm-avoiding algorithms
Huge data size not fitting into device memory	•Distributed computing •Streaming algorithms
Data coming in time	•Data buffering •Streaming algorithms
Non-homogeneous data	•Categorical→Numeric (counters, histograms, etc) •Homogeneous numeric data kernels • Conversions, Indexing, Repacking
Sparse/Missing/Noisy data	•Sparse data algorithms •Recovery methods (bootstrapping, outlier correction)

Distributed Computing



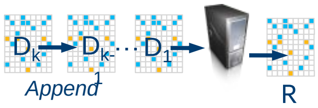
Streaming Computing



$S_{i+1} = T(S_i,D_i)$

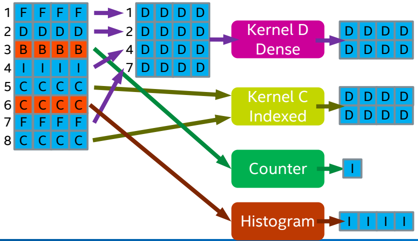
$R_{i+1} = F(S_{i+1})$

Offline Computing

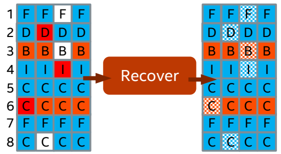


$R = F(D_1,\ldots,D_k)$

Converts, Indexing, Repacking



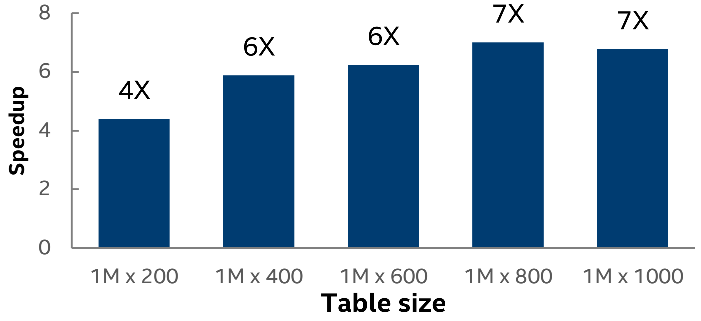
Data Recovery



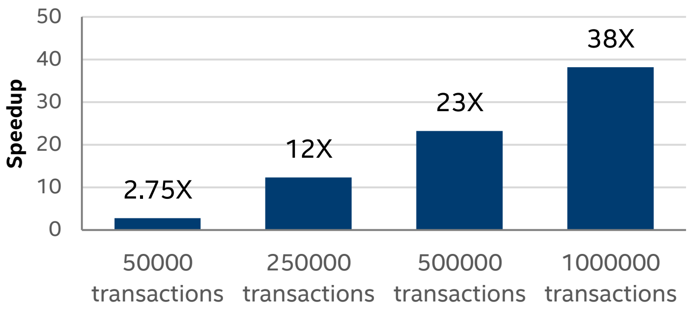
Intel® DAAL – Essentials for End-To-End Analytics

- C++ and Java*/Scala* library for data analytics
  - Targeting Python and R interfaces in future releases
  - “MKL” for machine learning with a few key differences
    - Optimizes **entire data flow** vs. compute part only
    - Targets both **data center** and **edge devices**
  - Supports offline, **online** and **distributed** data processing
  - Abstracted from cross-device communication layer
    - Allows plugging in different Big Data & IoT analytics frameworks
    - Comes with samples for Hadoop*, Spark*, MPI*
- Builds upon MKL/IPP for best performance

PCA (correlation method) on an 8-node Hadoop* cluster based on Intel® Xeon® Processors E5-2697 v3



Apriori on Intel® Xeon® Processor E5-2699 v3



Configuration Info - Versions: Intel® Data Analytics Acceleration Library 2016, CDH v5.3.1, Apache Spark* v1.2.0, Weka 3.6.12; Hardware: Intel® Xeon® Processor E5-2699 v3, 2 Eighteen-core CPUs (45MB LLC, 2.3GHz), 128GB of RAM per node; Operating System: CentOS 6.6 x86_64.

Summary

- Python is among top productivity languages
- Python is unfriendly to modern hardware, and hence to production use
- New tools and libraries allow making tradeoff between productivity and performance
- Intel is investing in Python to be more usable in prototyping and production
  - Intel® Distribution for Python - in Tech Preview Now!
  - Intel® VTune Amplifier for Python – available for evaluation Now!
- Big Data analytics requires end-to-end solutions
- Intel has "end-to-end" response for new analytics challenges
  - Intel® Data Analytics Acceleration Library 2016 product available Now!

Legal Disclaimer & Optimization Notice

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

experience what's inside™