
Table of Contents

| | |
|---|-------------|
| Foreword | xi |
| Preface | xiii |
| 1. Understanding Performant Python | 1 |
| The Fundamental Computer System | 2 |
| Computing Units | 2 |
| Memory Units | 5 |
| Communications Layers | 8 |
| Putting the Fundamental Elements Together | 10 |
| Idealized Computing Versus the Python Virtual Machine | 10 |
| So Why Use Python? | 14 |
| How to Be a Highly Performant Programmer | 16 |
| Good Working Practices | 17 |
| Some Thoughts on Good Notebook Practice | 19 |
| Getting the Joy Back into Your Work | 20 |
| 2. Profiling to Find Bottlenecks | 21 |
| Profiling Efficiently | 22 |
| Introducing the Julia Set | 23 |
| Calculating the Full Julia Set | 26 |
| Simple Approaches to Timing—print and a Decorator | 30 |
| Simple Timing Using the Unix time Command | 33 |
| Using the cProfile Module | 35 |
| Visualizing cProfile Output with SnakeViz | 39 |
| Using line_profiler for Line-by-Line Measurements | 40 |
| Using memory_profiler to Diagnose Memory Usage | 46 |
| Introspecting an Existing Process with PySpy | 54 |

| | |
|--|------------|
| Bytecode: Under the Hood | 55 |
| Using the dis Module to Examine CPython Bytecode | 55 |
| Different Approaches, Different Complexity | 57 |
| Unit Testing During Optimization to Maintain Correctness | 59 |
| No-op @profile Decorator | 60 |
| Strategies to Profile Your Code Successfully | 62 |
| Wrap-Up | 64 |
| 3. Lists and Tuples..... | 65 |
| A More Efficient Search | 68 |
| Lists Versus Tuples | 71 |
| Lists as Dynamic Arrays | 72 |
| Tuples as Static Arrays | 76 |
| Wrap-Up | 77 |
| 4. Dictionaries and Sets..... | 79 |
| How Do Dictionaries and Sets Work? | 83 |
| Inserting and Retrieving | 83 |
| Deletion | 87 |
| Resizing | 87 |
| Hash Functions and Entropy | 88 |
| Dictionaries and Namespaces | 92 |
| Wrap-Up | 95 |
| 5. Iterators and Generators..... | 97 |
| Iterators for Infinite Series | 101 |
| Lazy Generator Evaluation | 103 |
| Wrap-Up | 107 |
| 6. Matrix and Vector Computation..... | 109 |
| Introduction to the Problem | 110 |
| Aren't Python Lists Good Enough? | 115 |
| Problems with Allocating Too Much | 117 |
| Memory Fragmentation | 120 |
| Understanding perf | 122 |
| Making Decisions with perf's Output | 125 |
| Enter numpy | 126 |
| Applying numpy to the Diffusion Problem | 129 |
| Memory Allocations and In-Place Operations | 133 |
| Selective Optimizations: Finding What Needs to Be Fixed | 137 |
| numexpr: Making In-Place Operations Faster and Easier | 140 |
| A Cautionary Tale: Verify "Optimizations" (scipy) | 142 |

| | |
|--|------------|
| Lessons from Matrix Optimizations | 143 |
| Pandas | 146 |
| Pandas's Internal Model | 146 |
| Applying a Function to Many Rows of Data | 148 |
| Building DataFrames and Series from Partial Results Rather than Concatenating | 156 |
| There's More Than One (and Possibly a Faster) Way to Do a Job | 157 |
| Advice for Effective Pandas Development | 159 |
| Wrap-Up | 160 |
| 7. Compiling to C..... | 161 |
| What Sort of Speed Gains Are Possible? | 162 |
| JIT Versus AOT Compilers | 164 |
| Why Does Type Information Help the Code Run Faster? | 164 |
| Using a C Compiler | 165 |
| Reviewing the Julia Set Example | 166 |
| Cython | 167 |
| Compiling a Pure Python Version Using Cython | 167 |
| pyximport | 169 |
| Cython Annotations to Analyze a Block of Code | 170 |
| Adding Some Type Annotations | 172 |
| Cython and numpy | 176 |
| Parallelizing the Solution with OpenMP on One Machine | 178 |
| Numba | 180 |
| Numba to Compile NumPy for Pandas | 182 |
| PyPy | 183 |
| Garbage Collection Differences | 184 |
| Running PyPy and Installing Modules | 185 |
| A Summary of Speed Improvements | 186 |
| When to Use Each Technology | 187 |
| Other Upcoming Projects | 189 |
| Graphics Processing Units (GPUs) | 189 |
| Dynamic Graphs: PyTorch | 190 |
| Basic GPU Profiling | 193 |
| Performance Considerations of GPUs | 194 |
| When to Use GPUs | 196 |
| Foreign Function Interfaces | 197 |
| ctypes | 199 |
| cffi | 201 |
| f2py | 204 |
| CPython Module | 207 |
| Wrap-Up | 211 |

| | |
|--|------------|
| 8. Asynchronous I/O..... | 213 |
| Introduction to Asynchronous Programming | 215 |
| How Does async/await Work? | 218 |
| Serial Crawler | 219 |
| Gevent | 221 |
| tornado | 226 |
| aiohttp | 229 |
| Shared CPU-I/O Workload | 233 |
| Serial | 233 |
| Batched Results | 235 |
| Full Async | 238 |
| Wrap-Up | 243 |
| | |
| 9. The multiprocessing Module..... | 245 |
| An Overview of the multiprocessing Module | 248 |
| Estimating Pi Using the Monte Carlo Method | 250 |
| Estimating Pi Using Processes and Threads | 251 |
| Using Python Objects | 252 |
| Replacing multiprocessing with Joblib | 260 |
| Random Numbers in Parallel Systems | 263 |
| Using numpy | 264 |
| Finding Prime Numbers | 267 |
| Queues of Work | 273 |
| Verifying Primes Using Interprocess Communication | 278 |
| Serial Solution | 283 |
| Naive Pool Solution | 284 |
| A Less Naive Pool Solution | 285 |
| Using Manager.Value as a Flag | 286 |
| Using Redis as a Flag | 288 |
| Using RawValue as a Flag | 290 |
| Using mmap as a Flag | 291 |
| Using mmap as a Flag Redux | 293 |
| Sharing numpy Data with multiprocessing | 295 |
| Synchronizing File and Variable Access | 301 |
| File Locking | 302 |
| Locking a Value | 305 |
| Wrap-Up | 308 |
| | |
| 10. Clusters and Job Queues..... | 311 |
| Benefits of Clustering | 312 |
| Drawbacks of Clustering | 313 |
| \$462 Million Wall Street Loss Through Poor Cluster Upgrade Strategy | 315 |

| | |
|--|------------|
| Skype’s 24-Hour Global Outage | 315 |
| Common Cluster Designs | 316 |
| How to Start a Clustered Solution | 317 |
| Ways to Avoid Pain When Using Clusters | 318 |
| Two Clustering Solutions | 319 |
| Using IPython Parallel to Support Research | 319 |
| Parallel Pandas with Dask | 323 |
| NSQ for Robust Production Clustering | 326 |
| Queues | 327 |
| Pub/sub | 328 |
| Distributed Prime Calculation | 330 |
| Other Clustering Tools to Look At | 334 |
| Docker | 335 |
| Docker’s Performance | 335 |
| Advantages of Docker | 339 |
| Wrap-Up | 340 |
| 11. Using Less RAM..... | 341 |
| Objects for Primitives Are Expensive | 342 |
| The array Module Stores Many Primitive Objects Cheaply | 344 |
| Using Less RAM in NumPy with NumExpr | 346 |
| Understanding the RAM Used in a Collection | 350 |
| Bytes Versus Unicode | 352 |
| Efficiently Storing Lots of Text in RAM | 353 |
| Trying These Approaches on 11 Million Tokens | 354 |
| Modeling More Text with Scikit-Learn’s FeatureHasher | 362 |
| Introducing DictVectorizer and FeatureHasher | 362 |
| Comparing DictVectorizer and FeatureHasher on a Real Problem | 365 |
| SciPy’s Sparse Matrices | 367 |
| Tips for Using Less RAM | 370 |
| Probabilistic Data Structures | 371 |
| Very Approximate Counting with a 1-Byte Morris Counter | 372 |
| K-Minimum Values | 375 |
| Bloom Filters | 379 |
| LogLog Counter | 385 |
| Real-World Example | 389 |
| 12. Lessons from the Field..... | 393 |
| Streamlining Feature Engineering Pipelines with Feature-engine | 394 |
| Feature Engineering for Machine Learning | 394 |
| The Hard Task of Deploying Feature Engineering Pipelines | 395 |
| Leveraging the Power of Open Source Python Libraries | 395 |

| | |
|---|-----|
| Feature-engine Smooths Building and Deployment of Feature Engineering Pipelines | 396 |
| Helping with the Adoption of a New Open Source Package | 397 |
| Developing, Maintaining, and Encouraging Contribution to Open Source Libraries | 398 |
| Highly Performant Data Science Teams | 400 |
| How Long Will It Take? | 400 |
| Discovery and Planning | 401 |
| Managing Expectations and Delivery | 402 |
| Numba | 403 |
| A Simple Example | 404 |
| Best Practices and Recommendations | 405 |
| Getting Help | 409 |
| Optimizing Versus Thinking | 410 |
| Adaptive Lab’s Social Media Analytics (2014) | 412 |
| Python at Adaptive Lab | 413 |
| SoMA’s Design | 413 |
| Our Development Methodology | 414 |
| Maintaining SoMA | 414 |
| Advice for Fellow Engineers | 415 |
| Making Deep Learning Fly with RadimRehurek.com (2014) | 415 |
| The Sweet Spot | 416 |
| Lessons in Optimizing | 417 |
| Conclusion | 420 |
| Large-Scale Productionized Machine Learning at Lyst.com (2014) | 420 |
| Cluster Design | 421 |
| Code Evolution in a Fast-Moving Start-Up | 421 |
| Building the Recommendation Engine | 421 |
| Reporting and Monitoring | 422 |
| Some Advice | 422 |
| Large-Scale Social Media Analysis at Smesh (2014) | 423 |
| Python’s Role at Smesh | 423 |
| The Platform | 423 |
| High Performance Real-Time String Matching | 424 |
| Reporting, Monitoring, Debugging, and Deployment | 425 |
| PyPy for Successful Web and Data Processing Systems (2014) | 426 |
| Prerequisites | 427 |
| The Database | 428 |
| The Web Application | 428 |
| OCR and Translation | 429 |
| Task Distribution and Workers | 429 |
| Conclusion | 429 |

| | |
|--|------------|
| Task Queues at Lanyrd.com (2014) | 430 |
| Python's Role at Lanyrd | 430 |
| Making the Task Queue Performant | 431 |
| Reporting, Monitoring, Debugging, and Deployment | 431 |
| Advice to a Fellow Developer | 431 |
| Index..... | 433 |

