Ускорение производительности Python в 3.11

Дмитрий Анисов

Всем привет! Сегодня хочу поделиться с вами хорошими новостями, которые связаны с производительностью python в грядущем релизе 3.11 и то, что нас ожидает в будущем!

Достаточно долгое время существенных ускорений в CPython не наблюдалось, были определённые улучшения в некоторых последних версиях, но особо на итоговую производительность, даже в специальных задачах это не слишком сильно влияло, не говоря о задачах общего назначения, которые и составляют основную работу.

Всё начало меняться недавно, когда объявили о работе над проектом ***faster CPython***. Первая подготовительная работа уже началась в 3.10, а первые реальные улучшения должны были быть показаны в 3.11, что и произошло недавно при выходе новой альфа версии python 3.11.

Давайте, наконец, перейдём к самому главному и начнём сразу с основного тезиса, что нас ожидает в будущем:

> Python 3.11 is up to 10-60% faster than Python 3.10. On average, we measured a 1.22x speedup on the standard benchmark suite.
>
> This project focuses on two major areas in Python: faster startup and faster runtime.

Как мы видим, в грядущем релизе нас ожидает значительное ускорение производительности python! Конечно, в определённых задачах мы не увидим существенного различия(не берём в расчёт задачи, где основной bottleneck составляет i/o нагрузка), но сам факт того, что заявляются такие большие значения не может не радовать, и в особо узких местах(в определённых задачах) мы получим очень хороший прирост производительности, но уже на данный момент точно ясно одно, что общая итоговая производительность приложений вырастет для всех! А если у вас большой парк серверов и сервисов написанных на Python, которые там работают, то улучшение даже на 10 процентов вы заметите, не говоря уже о больших значениях, которые также заявляются.

Предлагаю на данном моменте ознакомиться с ростом производительности в синтетических бэнчмарках на текущий момент: клик

Почему это удалось? Что было сделано?

1) Faster Runtime

Была проведена работа с оптимизацией фреймов, которые создаются при вызове функций. А в большей части пользовательского кода, объекты фрейма, при вызове функций, теперь не создаются вовсе, что привело к итоговому приросту производительности на 3-7 процентов.

> *Python frames are created whenever Python calls a Python function. This frame holds execution information. The following are new frame optimizations:*
>
> *Streamlined the frame creation process.*
>
> *Avoided memory allocation by generously re-using frame space on the C stack.*
>
> *Streamlined the internal frame struct to contain only essential information. Frames previously held extra debugging and memory management information.*
>
> *Old-style frame objects are now created only when required by debuggers. For most user code, no frame objects are created at all. As a result, nearly all Python functions calls have sped up significantly. We measured a 3-7% speedup in pyperformance.*

2) Inlined Python function calls

Данная оптимизация позволяет полностью избежать вызова оценивающей функции C для интерпретации кода функций python, которая ранее вызывалась всегда. Для рекурсивных функций данная оптимизация позволила получить ускорение в 1.7 раза, а в общей производительности это улучшение принесло 1-3%.

> *During a Python function call, Python will call an evaluating C function to interpret that function's code. This effectively limits pure Python recursion to what's safe for the C stack.*
>
> *In 3.11, when CPython detects Python code calling another Python function, it sets up a new frame, and "jumps" to the new code inside the new frame. This avoids calling the C interpreting function altogether.*
>
> *Most Python function calls now consume no C stack space. This speeds up most of such calls. In simple recursive functions like fibonacci or factorial, a 1.7x speedup was observed. This also means recursive functions can recurse significantly deeper (if the user increases the recursion limit). We measured a 1-3% improvement in pyperforman*

И теперь самое главное!

3) Specializing Adaptive Interpreter

Это **основная и ключевая часть faster CPython** и причина почему python стал настолько быстрее. Идеи и решения, которые заложены в данной части, открывают новые возможности в росте производительности python.

Общая идея заключается в том, что несмотря на то, что python является динамическим языком, в большинстве программ есть области, в которых объекты и типы изменяются крайне редко, данная концепция известна как стабильность типов, и теперь во время выполнения python ищет такие места и заменяет операции на более специализированные. Это также приводит к ещё одной концепции, называемой встроенным кэшированием, теперь Python кэширует результаты ресурсоемких операций непосредственно в байт-коде.

Предлагаю самим ознакомиться с полным текстом, так как данная концепция крайне важна и является новой вехой в развитии python.

> *PEP 659 is one of the key parts of the faster CPython project. The general idea is that while Python is a dynamic language, most code has regions where objects and types rarely change. This concept is known as type stability.*
>
> *At runtime, Python will try to look for common patterns and type stability in the executing code. Python will then replace the current operation with a more specialized one. This specialized operation uses fast paths available only to those use cases/types, which generally outperform their generic counterparts. This also brings in another concept called inline caching, where Python caches the results of expensive operations directly in the bytecode.*
>
> *The specializer will also combine certain common instruction pairs into one superinstruction. This reduces the overhead during execution.*
>
> *Python will only specialize when it sees code that is "hot" (executed multiple times). This prevents Python from wasting time for run-once code. Python can also de-specialize when code is too dynamic or when the use changes. Specialization is attempted periodically, and specialization attempts are not too expensive. This allows specialization to adapt to new circumstances.*

Так же добавили ещё дополнительных оптимизаций, которые являются не такими важными, но тоже ускоряют производительность в определённых местах. Наиболее важными, на мой взгляд, являются следующие:

1. Objects now require less memory due to lazily created object namespaces. Their namespace dictionaries now also share keys more freely.
2. A more concise representation of exceptions in the interpreter reduced the time required for catching an exception by about 10%.
3. "Zero-cost" exceptions are implemented. The cost of try statements is almost eliminated when no exception is raised.
4. re's regular expression matching engine has been partially refactored, and now uses computed gotos (or "threaded code") on supported platforms. As a result, Python 3.11 executes the pyperformance regular expression benchmarks up to 10% faster than Python 3.10.

Почему это стало возможным?

Несмотря на то, что ещё недавно основные разработчики не намерены были ускорять CPython, так как считали, что текущей производительности достаточно для большинства задач и важнее развивать функционал языка, а в случае чего у нас есть Cython/ PyPy/Numba/мурус, расширения написанные на C и т.д.. То на данный момент всё поменялось.

1. Самое главное - это финансирование Microsoft, которое позволило работать на постоянной основе над проектом.
2. Найм на постоянную работу над CPython разработчика Марка Шэннона, ранее штатных разработчиков у CPython не было и язык развивался только благодаря сообществу и внеурочной работе её членов.
3. Использование наработок из HotPy, HotPy2.
4. Большой запрос среди сообщества на ускорение производительности python.
5. Конкуренция с другими языками, на которые приходилось мигрировать в связи с большой нагрузкой в определённых сервисах и "горячих местах".
6. Тренд на ускорение производительности и оптимизации в языках. На данный момент большое количество языков постоянно заявляют об улучшениях производительности и о готовности работать в данном направлении.
7. И, конечно же, желание основных разработчиков этим заниматься.

Какие изменения нас ждут в будущем? И какая конечная цель?

Общая цель — это ускорение CPython в **5** раз в ближайшие 4 года! Планируется это сделать в 4 отдельных этапа, каждый из которых увеличивает скорость CPython на **50** процентов.

Планируется улучшать интерпретатор, улучшать работу с памятью и самое главное добавить JIT компилятор, и далее постепенно его улучшать для создания превосходного машинного кода.

Конечно же будет сохранение обратной совместимости, что является немаловажной идеологией python на данный момент, после сложного перехода с python 2 на python 3.

С **подробным планом faster CPython** советую ознакомиться тут

И в конце предлагаю посмотреть никем особо не замеченное видео о том, как будет ускоряться python в будущем, почему это

происходит, какие изменения нас ждут и какие новые концепции появятся!

Источники:

1. What’s New In Python 3.11.
2. Implementation plan for speeding up CPython.
3. Making Python 5x FASTER with Guido van Rossum and Mark Shannon - Talk Python To Me.
4. Faster Python: Mark Shannon, author of newly endorsed plan, speaks to The Register.