[**Monte Carlo Method**](https://en.wikipedia.org/wiki/Monte_Carlo_method)**를 이용한 PI 연산 벤치마크 프로그램**

python은 간결한 문법을 사용하여 입문자들에게 쉽게 느껴지는 언어이지만, python의 약점 중 하나인 속도가 큰 걸림돌이 됩니다.

현재 python은 이러한 속도 문제를 해결하기 위하여 다른 언어들로 작성된 코드들을 [binding](https://en.wikipedia.org/wiki/Language_binding) 하여 실행시키는 방법을 주로 이용하며, 결과적으로 python은 [native code](https://www.techopedia.com/definition/8506/native-compiler)를 쉽게 호출하기 위한 인터페이스처럼 사용되고 있습니다.

이 프로젝트는 python에서 native code 또는 [AOT complied code](https://en.wikipedia.org/wiki/Ahead-of-time_compilation)를 호출하는 방법에 대하여 알아볼 것이고, 최종적으로 Monte Carlo Method를 이용한 PI 연산에 시간이 얼마나 걸리는지를 벤치마킹해볼 것입니다.

**벤치마킹 모델 구현**

먼저 벤치마킹에 사용할 Monte Carlo PI Compute(이하 MCPI) 함수가 필요한데, [이 글](https://fermium.tistory.com/324)의 구현을 참조하여 제작하였습니다.

//----------------

#mc\_pi\_python.py (line15~)

def mc\_pi(acc: int) -> float:

rng = fastrand()

hit\_count: int = 0

count: int = 0

for \_ in range(0, acc):

x: float = rng.rand()

y: float = rng.rand()

if x \* x + y \* y <= 1. :

hit\_count += 1

count += 1

return float(hit\_count) / float(count) \* 4.0

//----------------

MCPI는 random number generation이 핵심적으로 사용되는 알고리즘인데, CPython의 내장 모듈 random은 C binding으로 구현된것이기에, 벤치마크 형평성에 어긋나는 문제가 있습니다.

이를 해결하기 위하여 [xorshift64](https://en.wikipedia.org/wiki/Xorshift) 기반의 random number generator를 직접 구현하여 사용하였습니다 .

//----------------

#mc\_pi\_python.py (line2~13)

class fastrand:

def \_\_init\_\_(self, seed=0x956126898):

self.rctr = seed

def rand(self):

self.rctr %= 0xFFFFFFFF

self.rctr ^= (self.rctr << 13)

self.rctr %= 0xFFFFFFFF

self.rctr ^= (self.rctr >> 7)

self.rctr %= 0xFFFFFFFF

self.rctr ^= (self.rctr << 17)

self.rctr %= 0xFFFFFFFF

return float(self.rctr % 0xFFFFFFFF)/0xFFFFFFFF

//----------------

**벤치마킹 기능 구현**

python의 내장 모듈인 time을 이용하여 특정 함수의 수행 시간을 측정하는 것이 가능합니다.

//----------------

#main.py (line8~12)

def logging\_time(fn):

start\_time = time.time()

result = fn()

end\_time = time.time()

return (result, int((end\_time - start\_time) \* 1000))

//----------------

수행 시간을 측정한뒤에는 matplotlib를 사용하여 벤치마킹 결과를 그래프로 시각화합니다.

//----------------

#main.py (line31~39)

elapsedtimes.sort()

plt.bar(range(len(elapsedtimes)), [x[0] for x in elapsedtimes])

ax = plt.subplot()

ax.set\_xticks(range(len(elapsedtimes)))

ax.set\_xticklabels([x[1] for x in elapsedtimes])

plt.title(f"Monte Carlo Method PI Compute Benchmark\nIteration Count: {iterations}")

plt.xlabel("language")

plt.ylabel("elapsed time(ms)\nlower is better")

plt.show()

//----------------

위 과정을 거쳐서 벤치마킹을 위한 환경 준비는 전부 끝났습니다.

**Native Compiled Language Binding:** [**Rust**](https://en.wikipedia.org/wiki/Rust_(programming_language))

python에서 다른 언어로 작성된 코드를 built-in module로 사용하기 위해서는 [extending 규약](https://docs.python.org/3/extending/extending.html)을 따를 수 있는 C 스타일의 export가 가능한 언어가 일반적으로 요구됩니다.

Rust에서는 이에 맞게 자동으로 파이썬 바인딩을 해주는 도구인 [PyO3](https://github.com/PyO3/pyo3) 가 존재하여 이를 사용하여 먼저 Rust에서 몬테카를로 시뮬레이션을 구현하였습니다.

//----------------

//mc\_pi\_rust.py (line4~28)

struct Fastrand {

rctr: u64

}

impl Fastrand {

fn new() -> Fastrand {

Fastrand { rctr: 0x956126898 }

}

fn splitmix64(&mut self, seed: u64) {

self.rctr = seed;

}

fn rand(&mut self) -> f64 {

self.rctr %= 0xFFFFFFFF;

self.rctr ^= self.rctr.wrapping\_shl(13);

self.rctr %= 0xFFFFFFFF;

self.rctr ^= self.rctr.wrapping\_shr(7);

self.rctr %= 0xFFFFFFFF;

self.rctr ^= self.rctr.wrapping\_shl(17);

self.rctr %= 0xFFFFFFFF;

((self.rctr % 0xFFFFFFFF) as f64) / (0xFFFFFFFFu32 as f64)

//self.rctr as f64 / 0xFFFFFFFFFFFFFFFFu64 as f64

}

}

#[pyfunction]

fn mc\_pi(acc: i32) -> PyResult<f64> {

let mut rng = Fastrand::new();

let mut hit\_count = 0;

let mut count = 0;

for \_i in 0..acc {

let x: f64 = rng.rand();

let y: f64 = rng.rand();

if x \* x + y \* y <= 1. {

hit\_count += 1;

}

count += 1;

}

Ok(hit\_count as f64 / count as f64 \* 4.)

}

//----------------

**AOT Compiled Language Binding:** [**C#**](https://en.wikipedia.org/wiki/C_Sharp_(programming_language))

c# 은 [CLR](https://docs.microsoft.com/en-us/dotnet/standard/clr) 위에서 동작하는 언어이기 때문에 CLR을 로드하지 않으면 python에서 binding 하여 사용하는 것은 불가능합니다.

이를 해결하기 위하여 c# 코드를 c++으로 binding 한 뒤에 python으로 다시 binding 하는 방법을 채택하였습니다. 이 과정에서 c# binding 을 도와주는 도구인 [Unmanaged Exports](https://sites.google.com/site/robertgiesecke/Home/uploads/unmanagedexports)와 [DllExport](https://github.com/3F/DllExport)를 사용하였습니다.

c#에서도 동일한 사양의 알고리즘을 사용하여 MCPI를 구현하였습니다.

//----------------

//mc\_py\_csharp/McpiCsharp.cs (line8~43)

private class Fastrand

{

private ulong \_rctr;

public Fastrand() => \_rctr = 0x956126898;

public void SplitMix64(ulong seed) => \_rctr = seed;

public double Rand()

{

\_rctr %= 0xFFFFFFFF;

\_rctr ^= \_rctr << 13;

\_rctr %= 0xFFFFFFFF;

\_rctr ^= \_rctr >> 7;

\_rctr %= 0xFFFFFFFF;

\_rctr ^= \_rctr << 17;

\_rctr %= 0xFFFFFFFF;

return (float) (\_rctr % 0xFFFFFFFF) / 0xFFFFFFFF;

}

}

[DllExport("mc\_pi", CallingConvention.Cdecl)]

public static double Mcpi(int acc)

{

var rng = new Fastrand();

var hitCount = 0;

var count = 0;

for (var i = 0; i < acc; i++)

{

var x = rng.Rand();

var y = rng.Rand();

if (x \* x + y \* y <= 1.0) {

hitCount += 1;

}

count += 1;

}

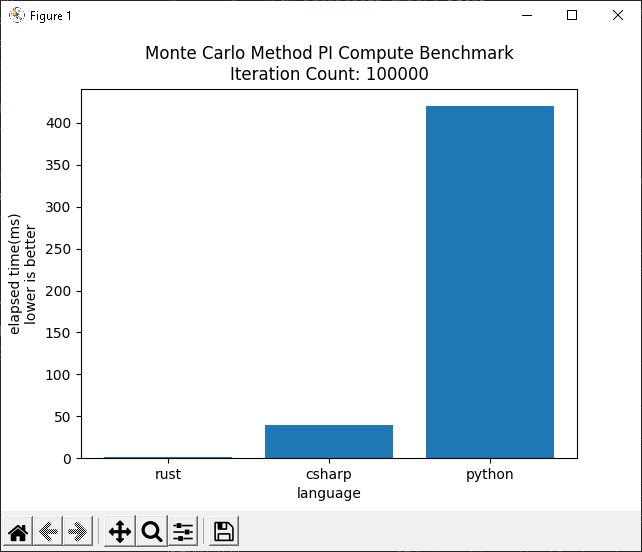
return (float)hitCount / count \* 4.0;

}

//----------------

**벤치마크 결과**

10만번 이터레이션한 벤치마킹 실행결과 입니다.



소요시간 입니다.

|  |  |
| --- | --- |
| rust | 1ms |
| c# | 40 ms |
| python | 426ms |

native code 를 생성하는 rust의 소요시간이 가장 빠르게 나왔으며, 그 뒤로 AOT compile 되는 c#, 마지막으로 스크립트 언어인 python 의 순서대로 결과가 나왔습니다.

비고

* [프로젝트 레포지토리](https://github.com/noname0310/PythonBindingSample)가 존재하며 벤치마킹에 사용된 모든 코드가 포함되어있습니다.
* 각 플랫폼별 사용된 도구의 버전입니다.   
  [CPython 3.9.5](https://www.python.org/downloads/release/python-395/),   
  [rustc 1.52.0](https://blog.rust-lang.org/2021/05/06/Rust-1.52.0.html), [PyO3 0.13.2](https://docs.rs/crate/pyo3/0.13.2),   
  [.NET 5.0](https://dotnet.microsoft.com/download/dotnet/5.0), [UnmanagedExports.Repack 1.0.4](https://www.nuget.org/packages/UnmanagedExports.Repack/), [DllExport 1.7.4](https://www.nuget.org/packages/DllExport/)
* rust 코드에 사용된 빌드 플래그 입니다. --lib --manifest-path Cargo.toml --features pyo3/extension-module --target x86\_64-pc-windows-msvc --release --verbose -- --crate-type cdylib
* C# 코드는 x64 Release 로 빌드되었습니다.   
  빌드 옵션은 VS Community 2019 Preview 16.11.0 Preview 1.0 의 기본값을 사용하였습니다.