p2rng – A C++ Parallel Random Number Generator Library for the Masses

Armin Sobhani

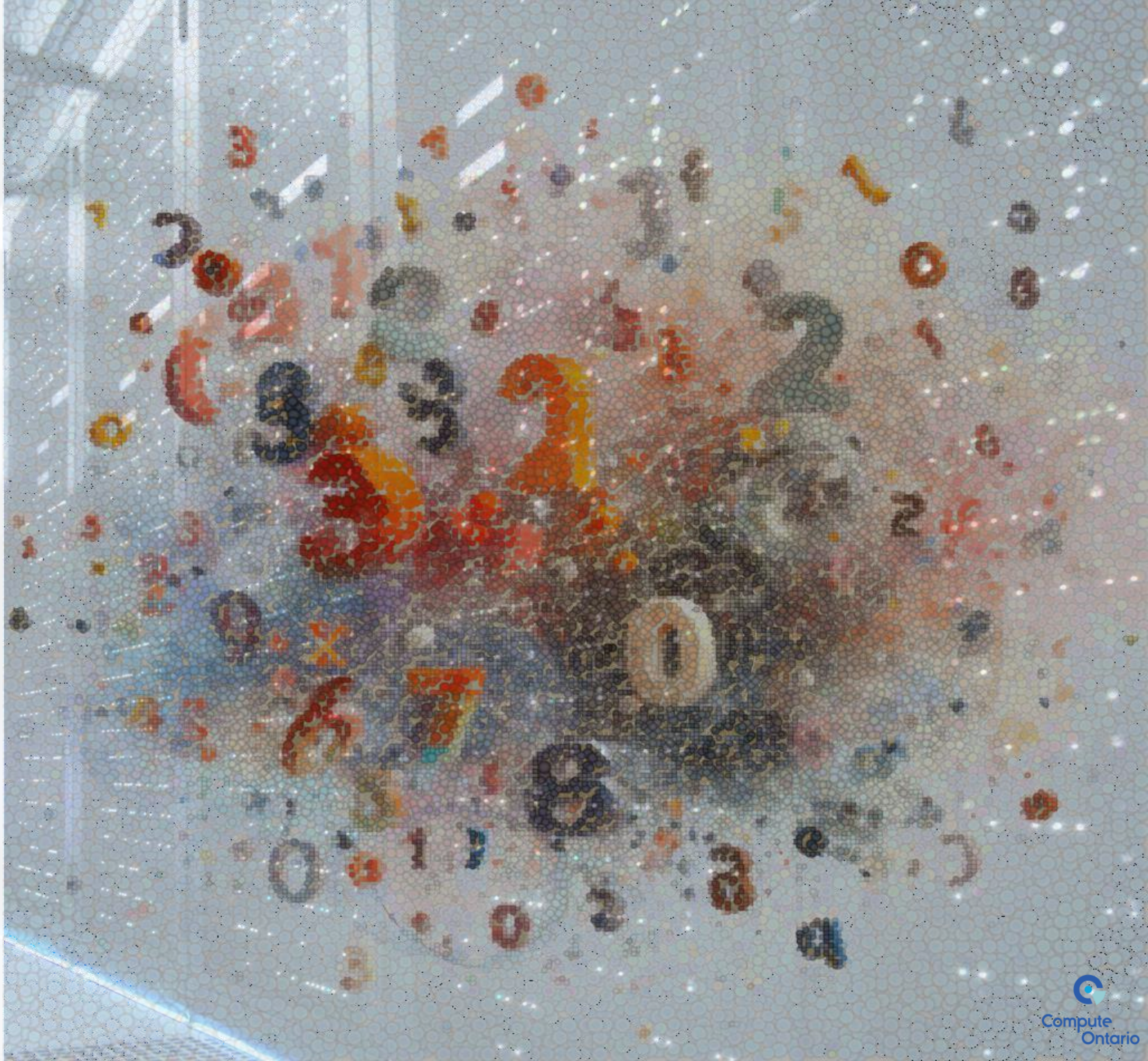
asobhani@sharcnet.ca

https://staff.sharcnet.ca/asobhani

SHARCNET | Compute Ontario

HPC Technical Consultant

OntarioTech UNIVERSITY

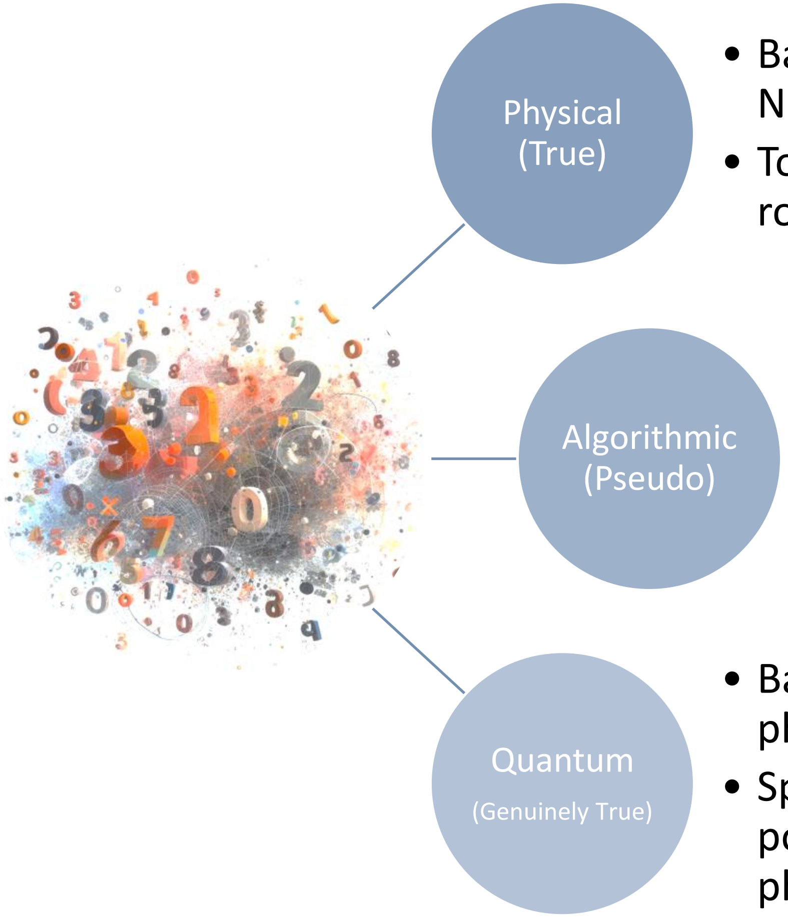


SHARCNET™

Compute Ontario

Preliminary Concepts

Types of Randomness



Physical
(True)

- Based on Newtonian physics
- Tossing a coin, rolling dice, etc.

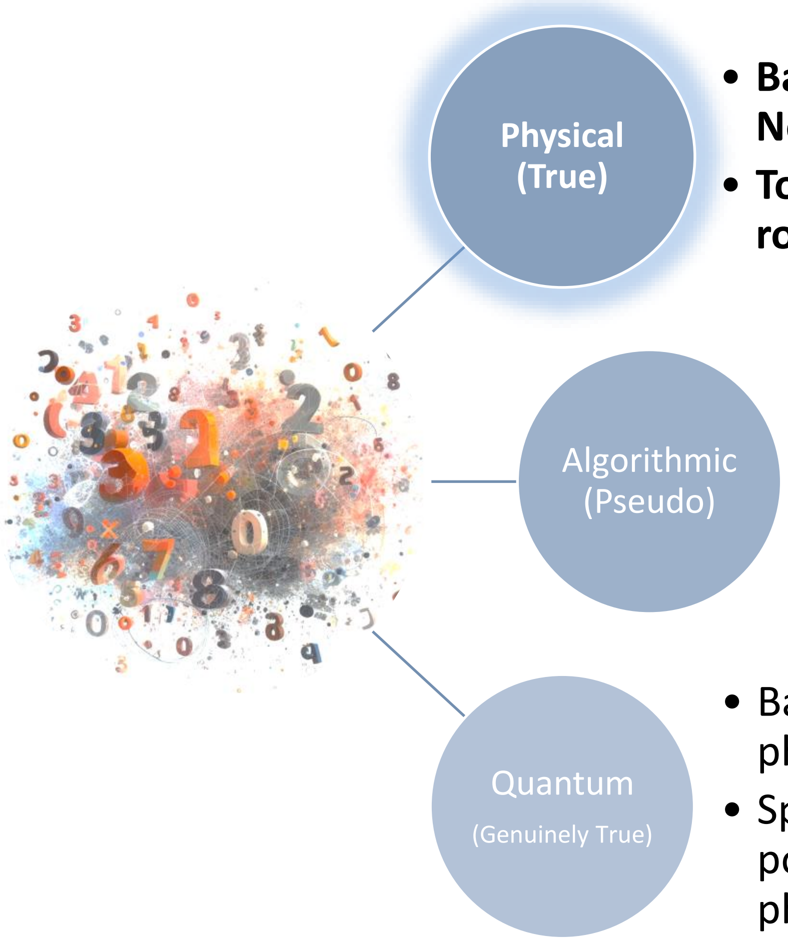
Algorithmic
(Pseudo)

- Based on a computer algorithm
- LCG, mt19937, etc.

Quantum
(Genuinely True)

- Based on quantum physics
- Spin of an electron, polarization of a photon

Types of Randomness



Physical (True)

- **Based on Newtonian physics**
- **Tossing a coin, rolling dice, etc.**

Algorithmic (Pseudo)

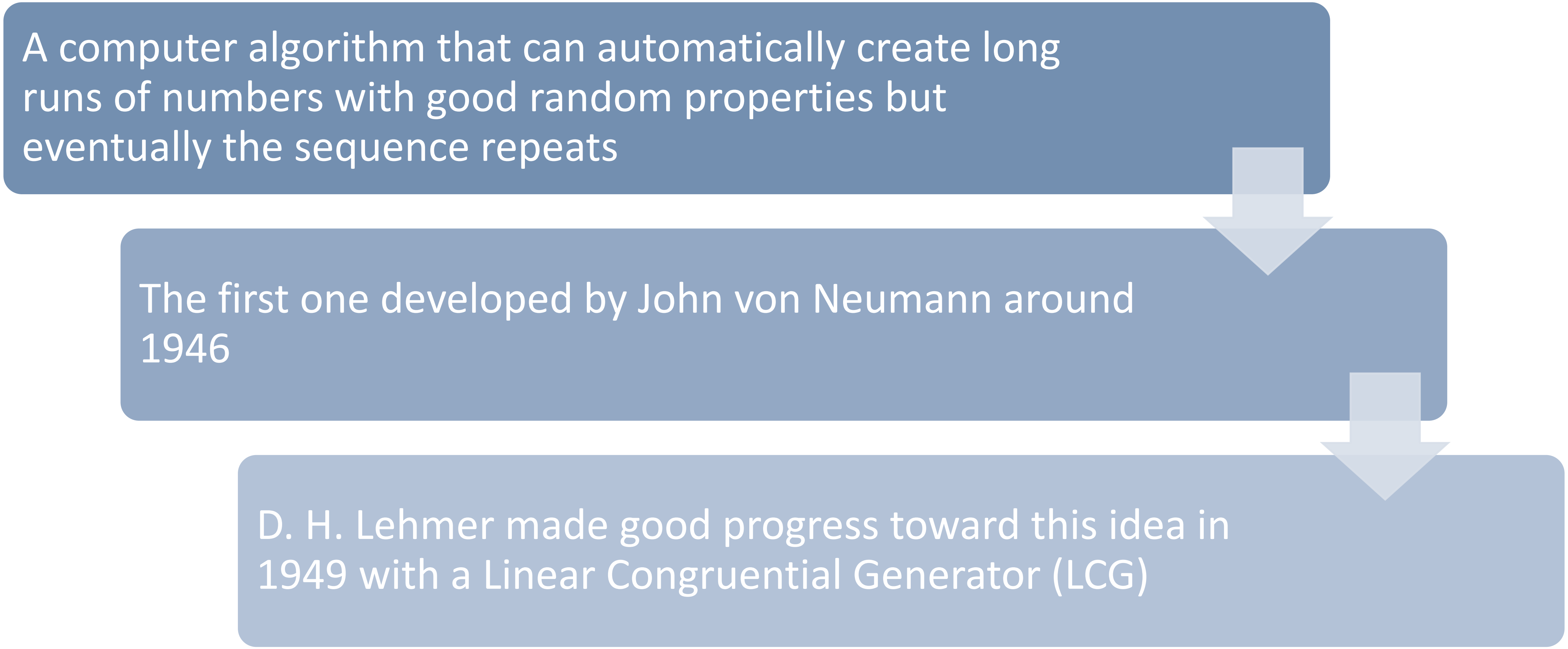
- Based on a computer algorithm
- LCG, mt19937, etc.

Quantum (Genuinely True)

- Based on quantum physics
- Spin of an electron, polarization of a photon

DOI: 10.48550/arxiv.2310.04153

Algorithmic Random Number Generators (RNG)



A computer algorithm that can automatically create long runs of numbers with good random properties but eventually the sequence repeats

The first one developed by John von Neumann around 1946

D. H. Lehmer made good progress toward this idea in 1949 with a Linear Congruential Generator (LCG)

Important Attributes of an RNG

Output Bits

- Number of bits in the generated number

Period

- The smallest number of steps after which the generator starts repeating itself
- Usually expressed in power of two

Seed

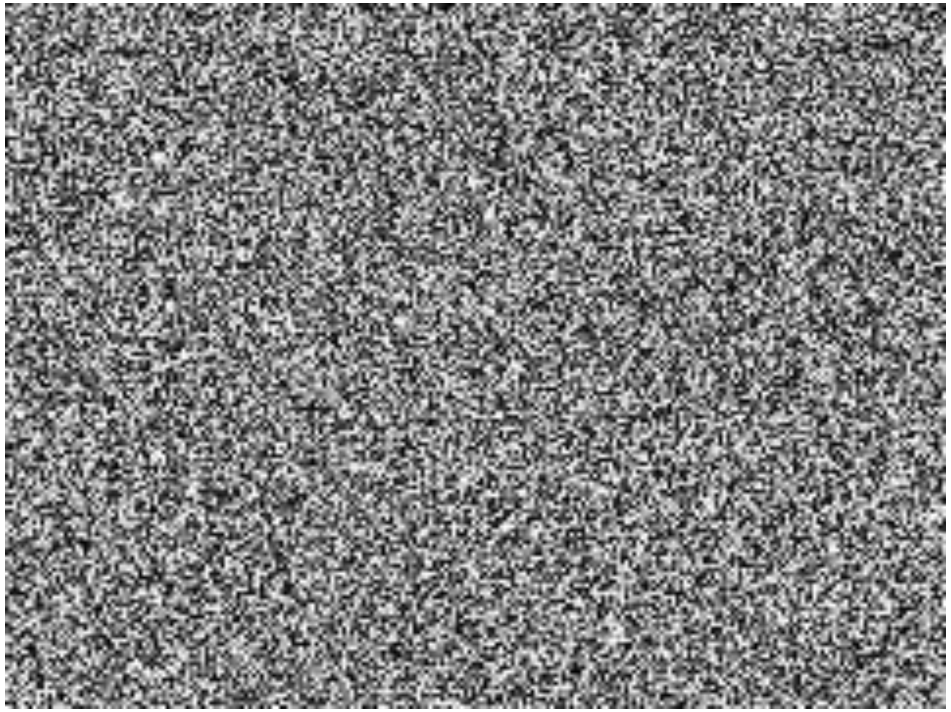
- An initial value that determines the sequence of random numbers generated by the algorithm

Footprint (AKA space usage)

- Size of the internal state in bytes

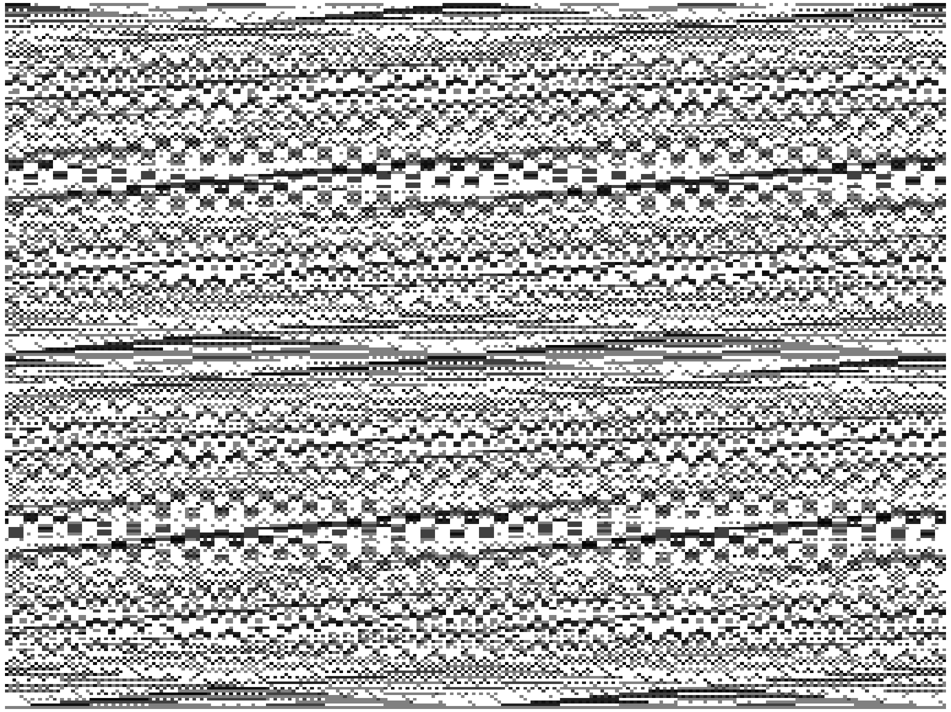
Statistical Testing of RNGs

Looking at Randomgrams



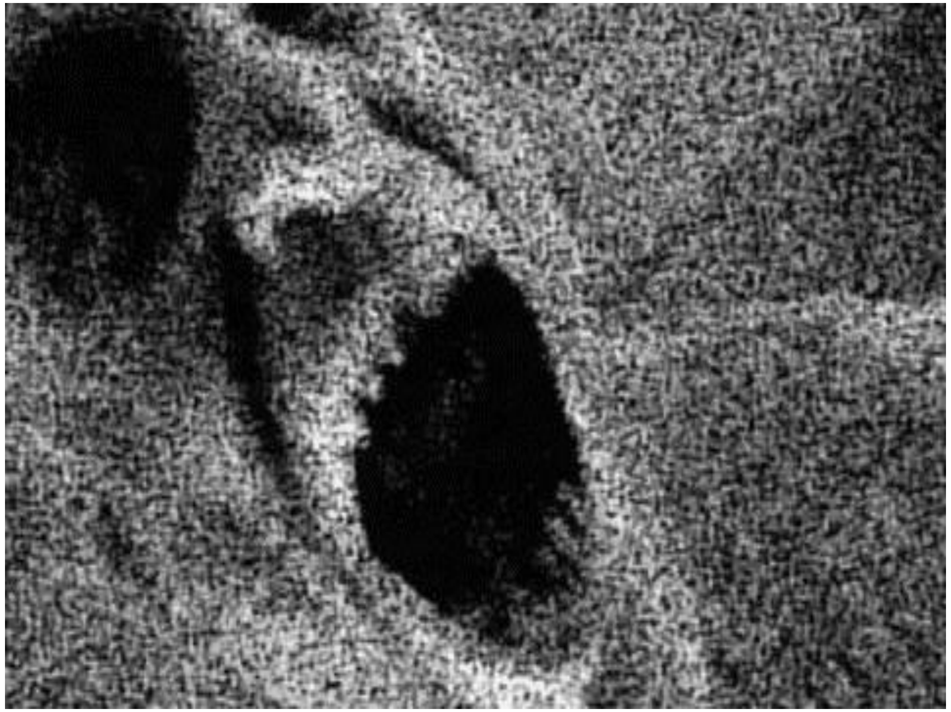
Statistical Testing of RNGs

Looking at Randomgrams

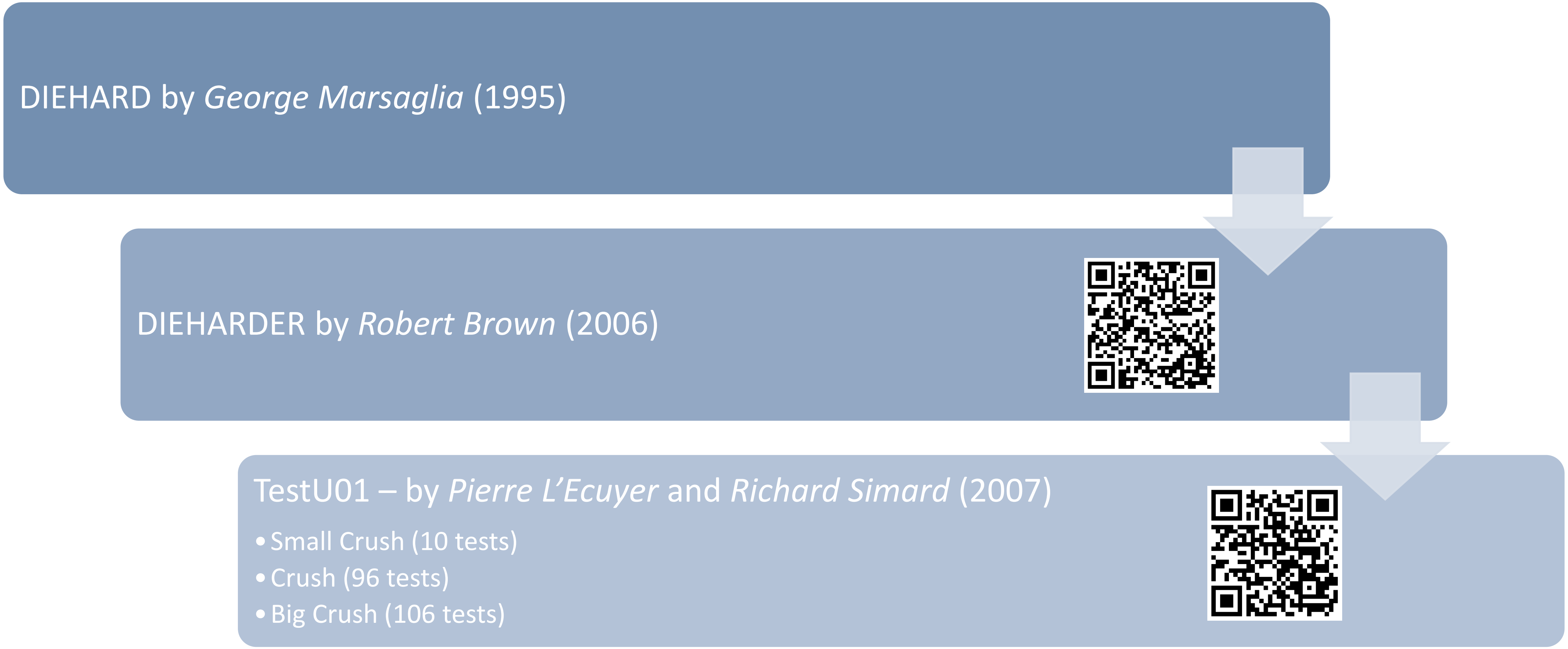


Statistical Testing of RNGs

Looking at Randomgrams



Batteries of Tests for RNGs



DIEHARD by *George Marsaglia* (1995)

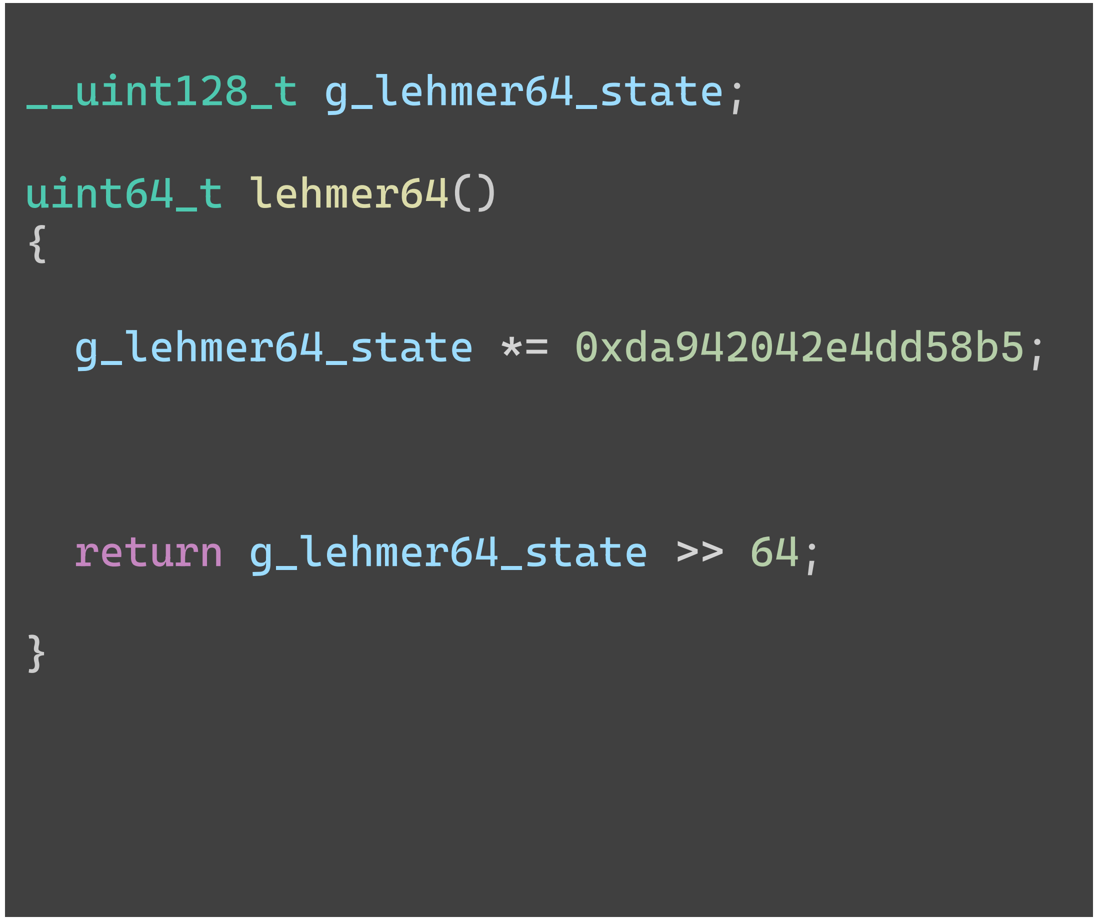
DIEHARDER by *Robert Brown* (2006)

TestU01 – by *Pierre L'Ecuyer* and *Richard Simard* (2007)

- Small Crush (10 tests)
- Crush (96 tests)
- Big Crush (106 tests)

Two Parts of an Algorithmic RNG

- The State-Transition Function
- The Output Function



```
__uint128_t g_lehmer64_state;

uint64_t lehmer64()
{
  g_lehmer64_state *= 0xda942042e4dd58b5;


  return g_lehmer64_state >> 64;
}
```

Two Parts of an Algorithmic RNG

The State-Transition Function

The Output Function

```
__uint128_t g_lehmer64_state;

uint64_t lehmer64()
{
  g_lehmer64_state *= 0xda942042e4dd58b5;


  return g_lehmer64_state >> 64;
}
```

Two Parts of an Algorithmic RNG

The State-Transition Function

The Output Function

```
__uint128_t g_lehmer64_state;

uint64_t lehmer64()
{

  g_lehmer64_state *= 0xda942042e4dd58b5;



  return g_lehmer64_state >> 64;

}
```

Reproducibility vs. Playing Fair

Reproducibility

Getting same sequence of numbers using the same seed and distribution

Refers to serial random number generation

Playing Fair

A parallel Monte Carlo simulation *plays fair*, when its outcome is strictly independent of the underlying hardware

Getting same sequence regardless of the number of parallel threads

Serial Random Number Generation

in C++

Before C++11

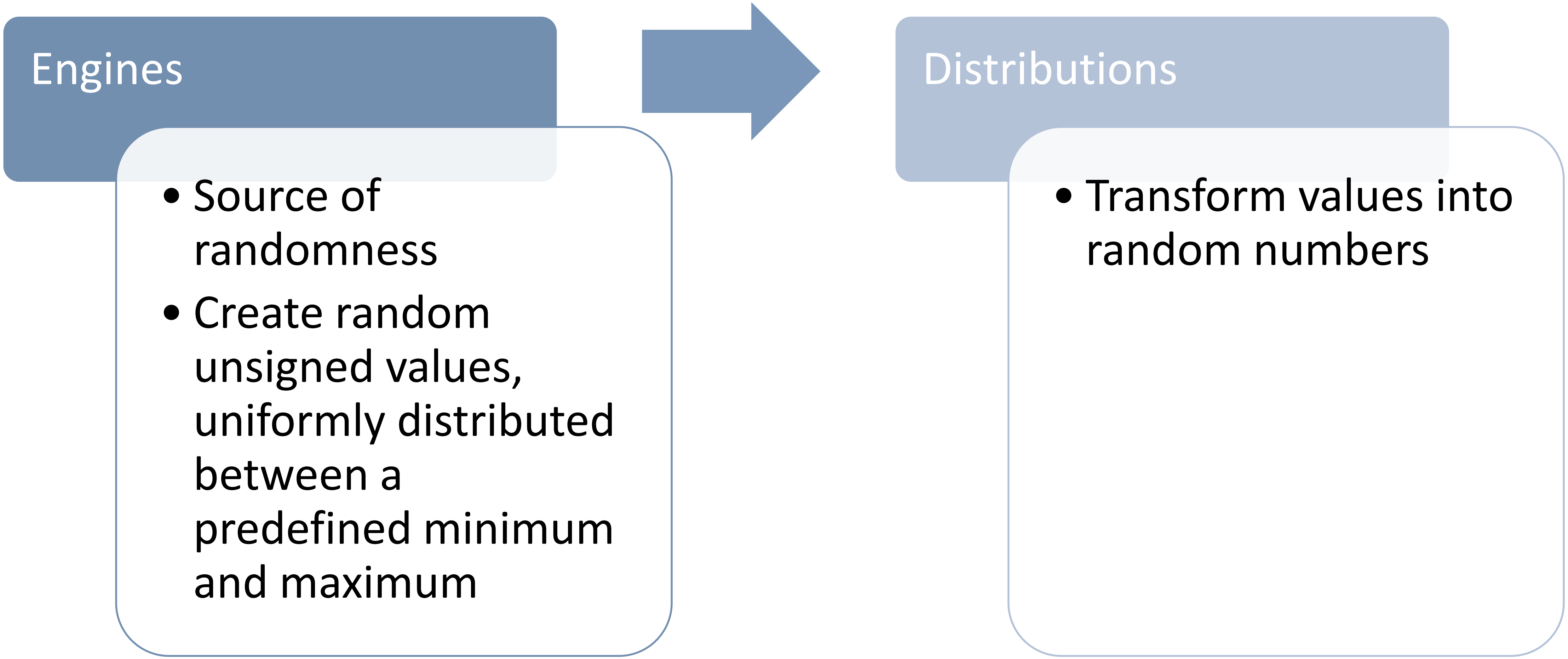
`srand()`

for seeding

`rand()`

for the next random number

Since C++11



Engines

- Source of randomness
- Create random unsigned values, uniformly distributed between a predefined minimum and maximum

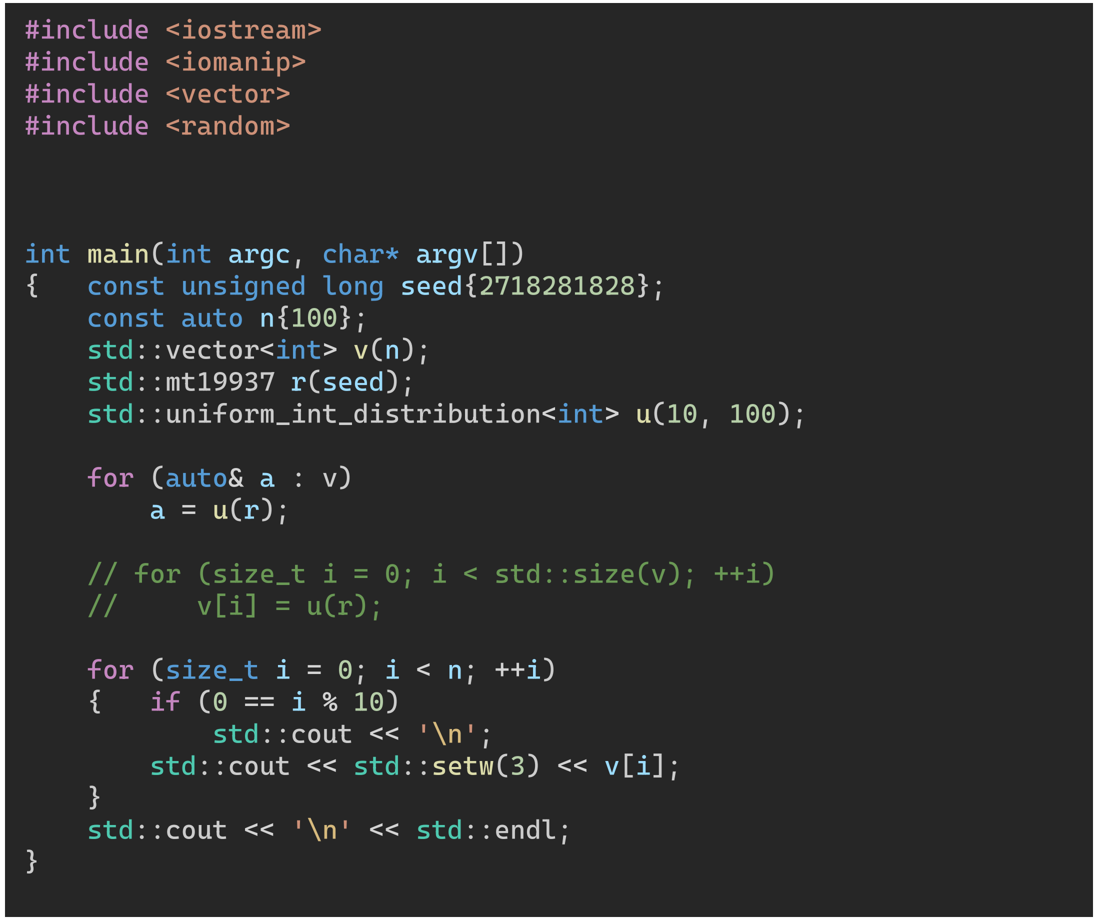
Distributions

- Transform values into random numbers

Check Paul’s Talk for More Information

https://youtu.be/tjP5juz3O1Q

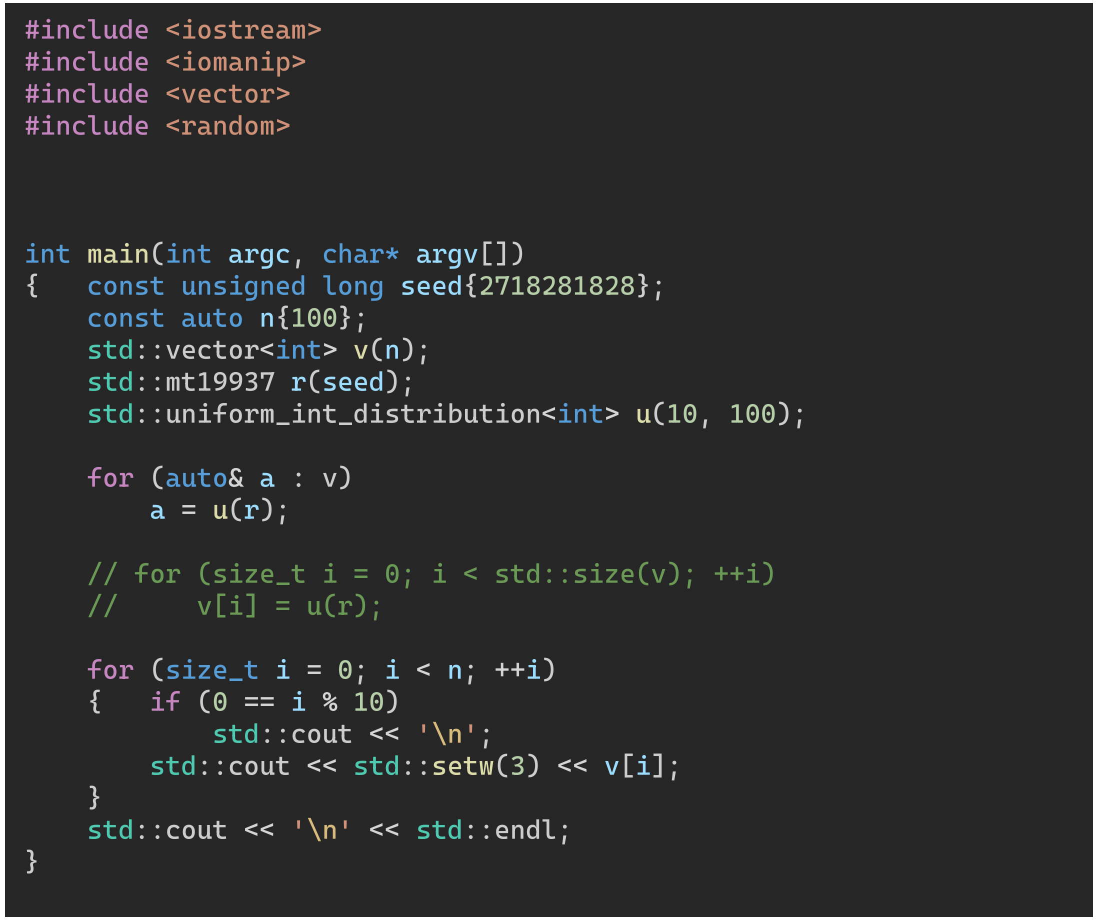
Using **for** Loops



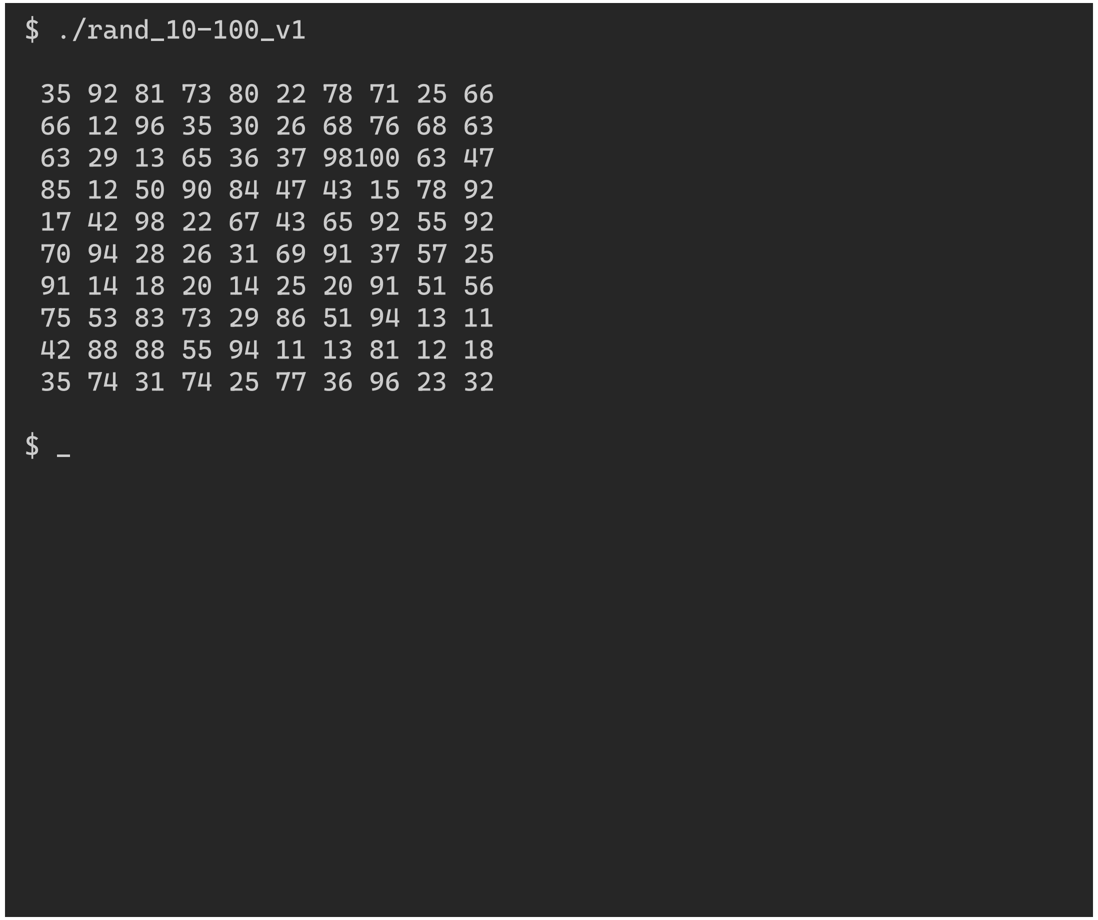
```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>



int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    for (auto& a : v)
        a = u(r);

    // for (size_t i = 0; i < std::size(v); ++i)
    //     v[i] = u(r);

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

```
$ _
```

Using **for** Loops



```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>



int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    for (auto& a : v)
        a = u(r);

    // for (size_t i = 0; i < std::size(v); ++i)
    //     v[i] = u(r);

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```



```
$ ./rand_10-100_v1

 35 92 81 73 80 22 78 71 25 66
 66 12 96 35 30 26 68 76 68 63
 63 29 13 65 36 37 98100 63 47
 85 12 50 90 84 47 43 15 78 92
 17 42 98 22 67 43 65 92 55 92
 70 94 28 26 31 69 91 37 57 25
 91 14 18 20 14 25 20 91 51 56
 75 53 83 73 29 86 51 94 13 11
 42 88 88 55 94 11 13 81 12 18
 35 74 31 74 25 77 36 96 23 32

$ _
```

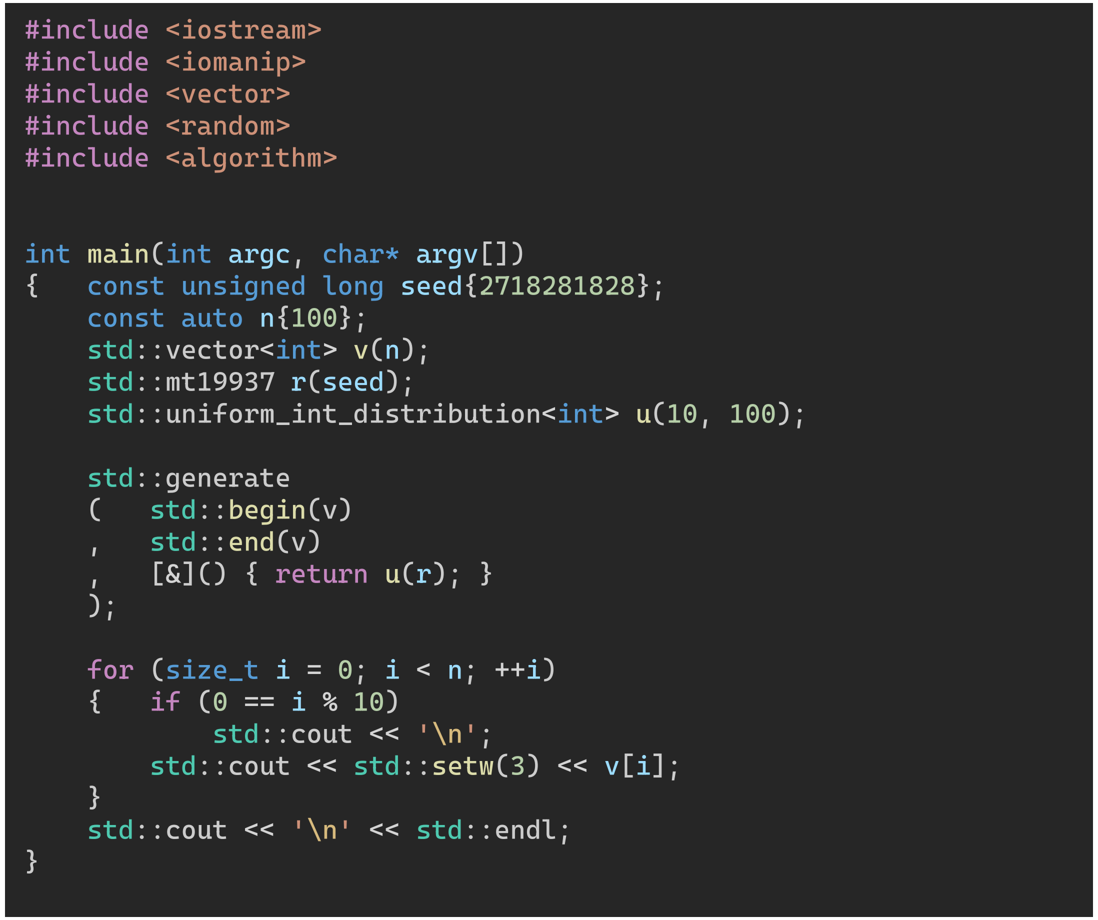
Using for Loops

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>



int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    for (auto& a : v)
        a = u(r);

    // for (size_t i = 0; i < std::size(v); ++i)
    //     v[i] = u(r);

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

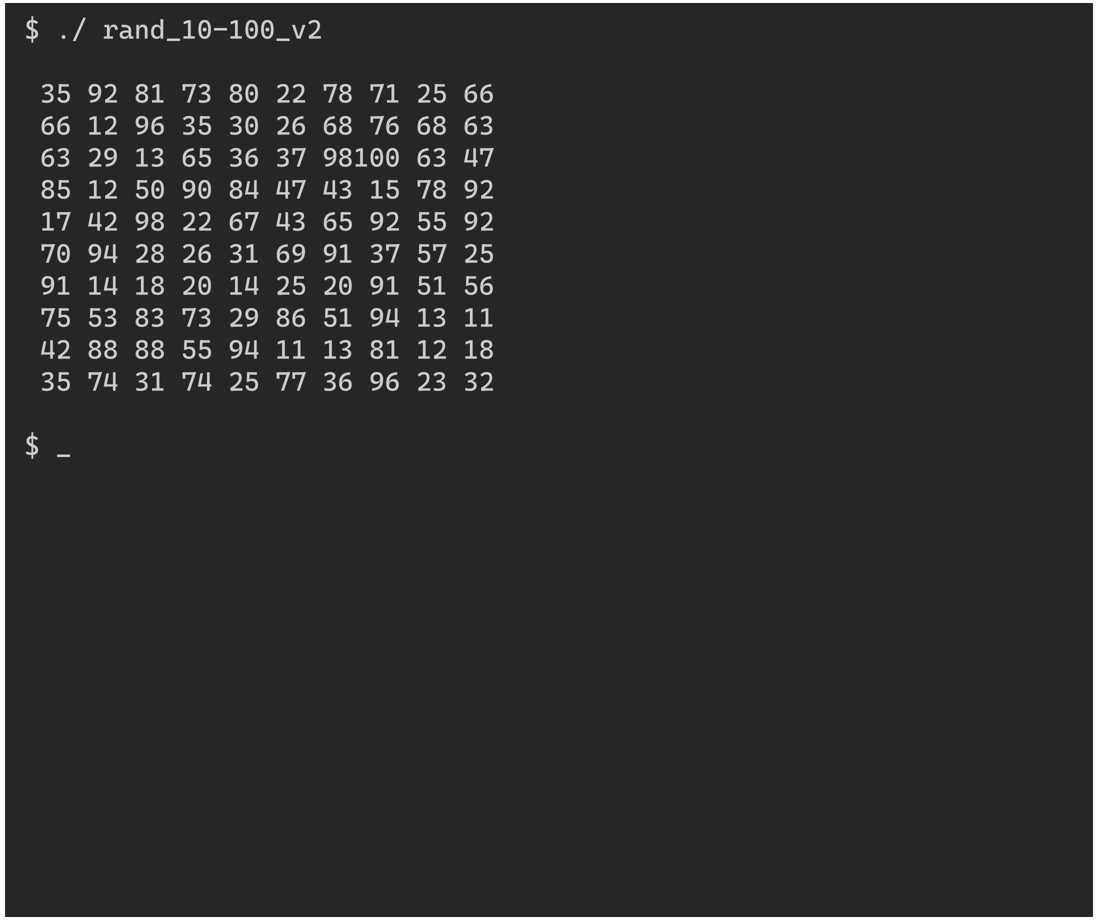
closed range: [10, 100]

```
$ ./rand_10-100_v1

 35 92 81 73 80 22 78 71 25 66
 66 12 96 35 30 26 68 76 68 63
 63 29 13 65 36 37 98100 63 47
 85 12 50 90 84 47 43 15 78 92
 17 42 98 22 67 43 65 92 55 92
 70 94 28 26 31 69 91 37 57 25
 91 14 18 20 14 25 20 91 51 56
 75 53 83 73 29 86 51 94 13 11
 42 88 88 55 94 11 13 81 12 18
 35 74 31 74 25 77 36 96 23 32

$ _
```

std:generate() with *Lambda*

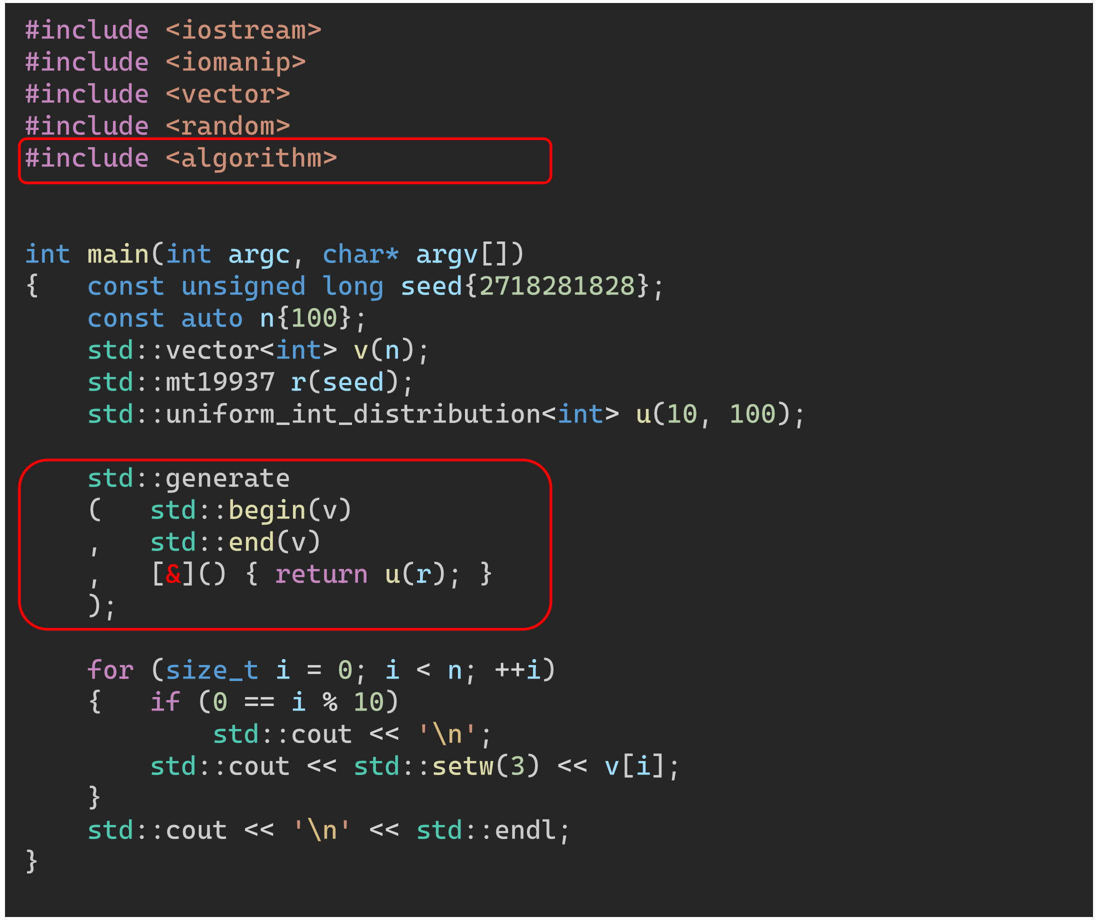


```
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>


int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    std::generate
    (   std::begin(v)
    ,   std::end(v)
    ,   [&]() { return u(r); }
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

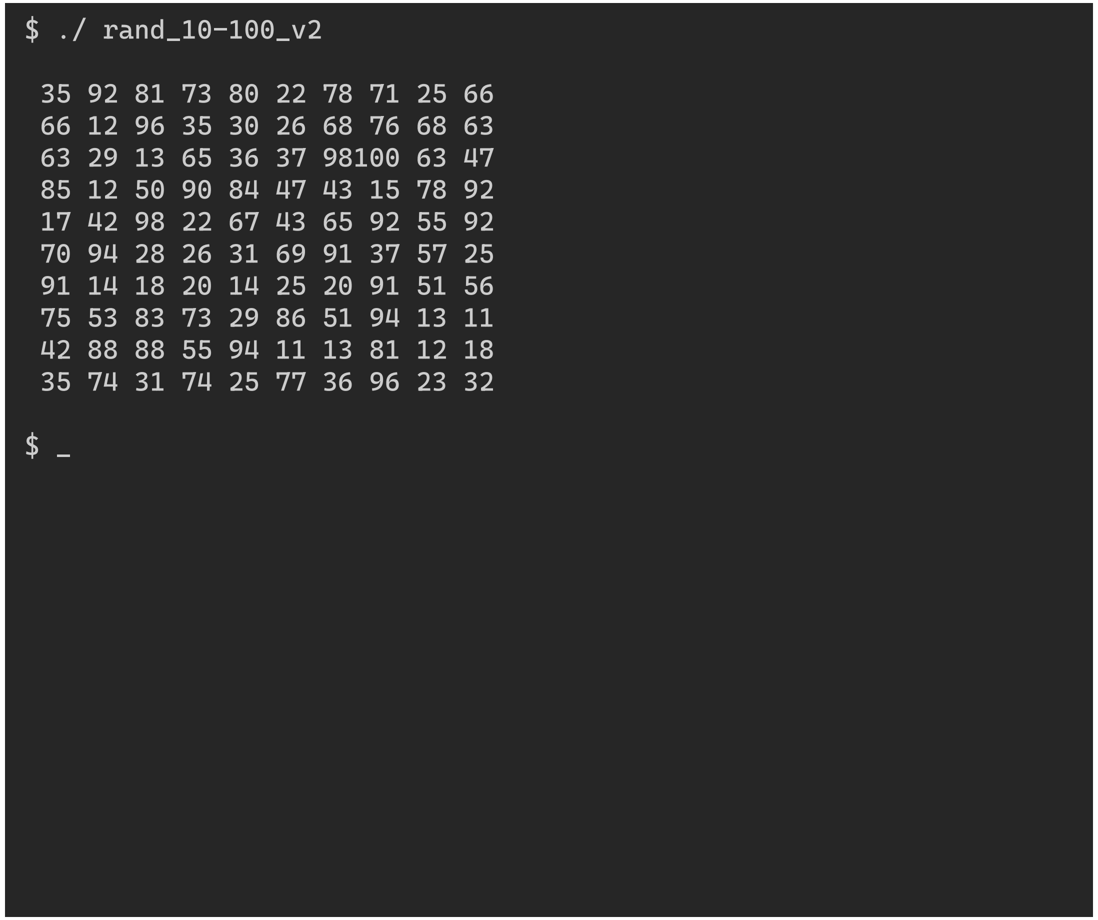


```
$ ./ rand_10-100_v2

 35 92 81 73 80 22 78 71 25 66
 66 12 96 35 30 26 68 76 68 63
 63 29 13 65 36 37 98100 63 47
 85 12 50 90 84 47 43 15 78 92
 17 42 98 22 67 43 65 92 55 92
 70 94 28 26 31 69 91 37 57 25
 91 14 18 20 14 25 20 91 51 56
 75 53 83 73 29 86 51 94 13 11
 42 88 88 55 94 11 13 81 12 18
 35 74 31 74 25 77 36 96 23 32

$ _
```

std:generate() with *Lambda*

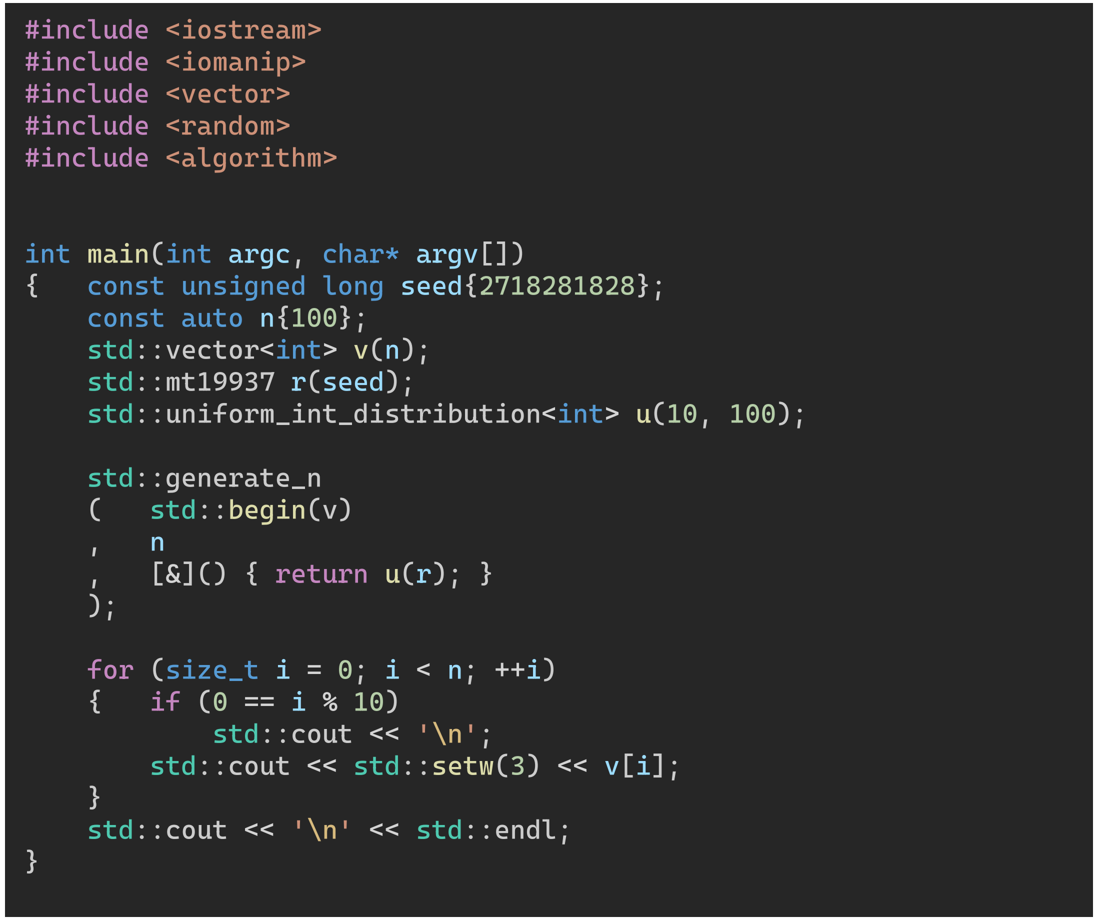


```
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>


int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    std::generate
    (   std::begin(v)
    ,   std::end(v)
    ,   [&]() { return u(r); }
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

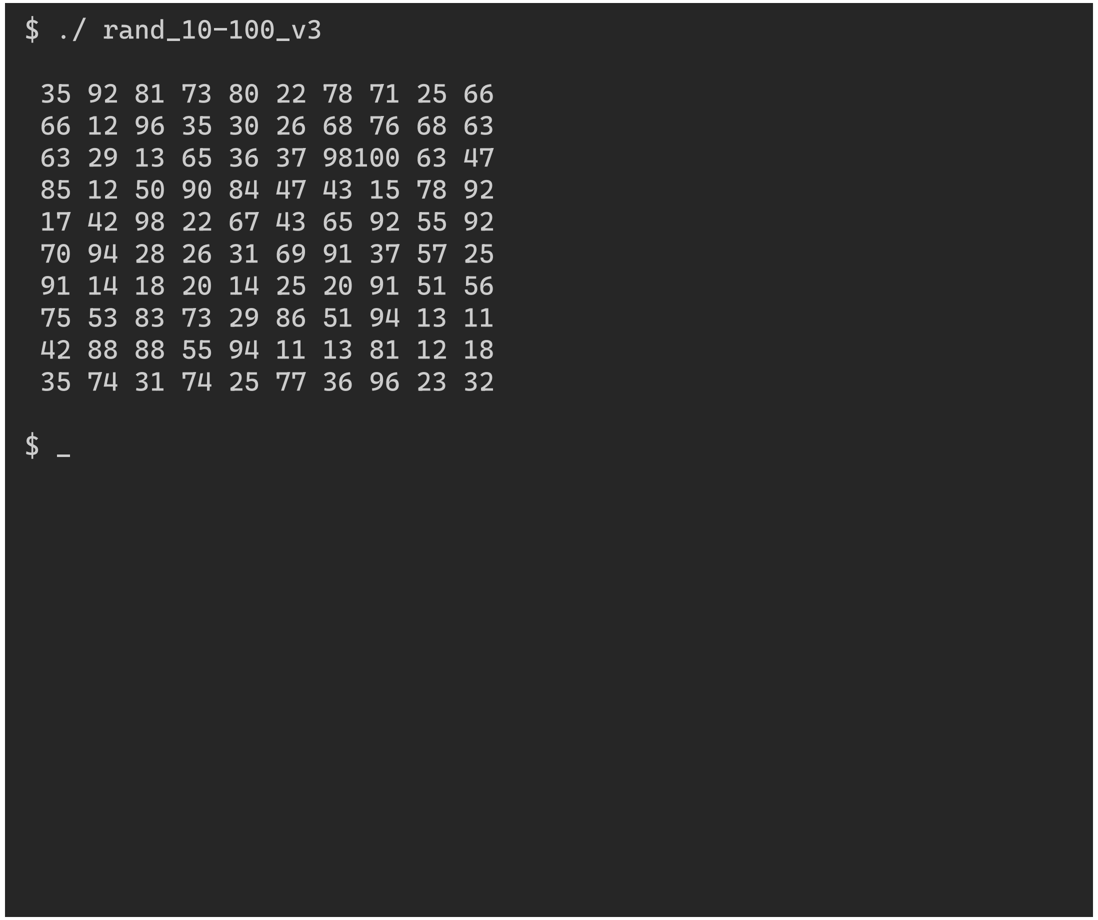


```
$ ./ rand_10-100_v2

 35 92 81 73 80 22 78 71 25 66
 66 12 96 35 30 26 68 76 68 63
 63 29 13 65 36 37 98100 63 47
 85 12 50 90 84 47 43 15 78 92
 17 42 98 22 67 43 65 92 55 92
 70 94 28 26 31 69 91 37 57 25
 91 14 18 20 14 25 20 91 51 56
 75 53 83 73 29 86 51 94 13 11
 42 88 88 55 94 11 13 81 12 18
 35 74 31 74 25 77 36 96 23 32

$ _
```

std:generate_n() with *Lambda*

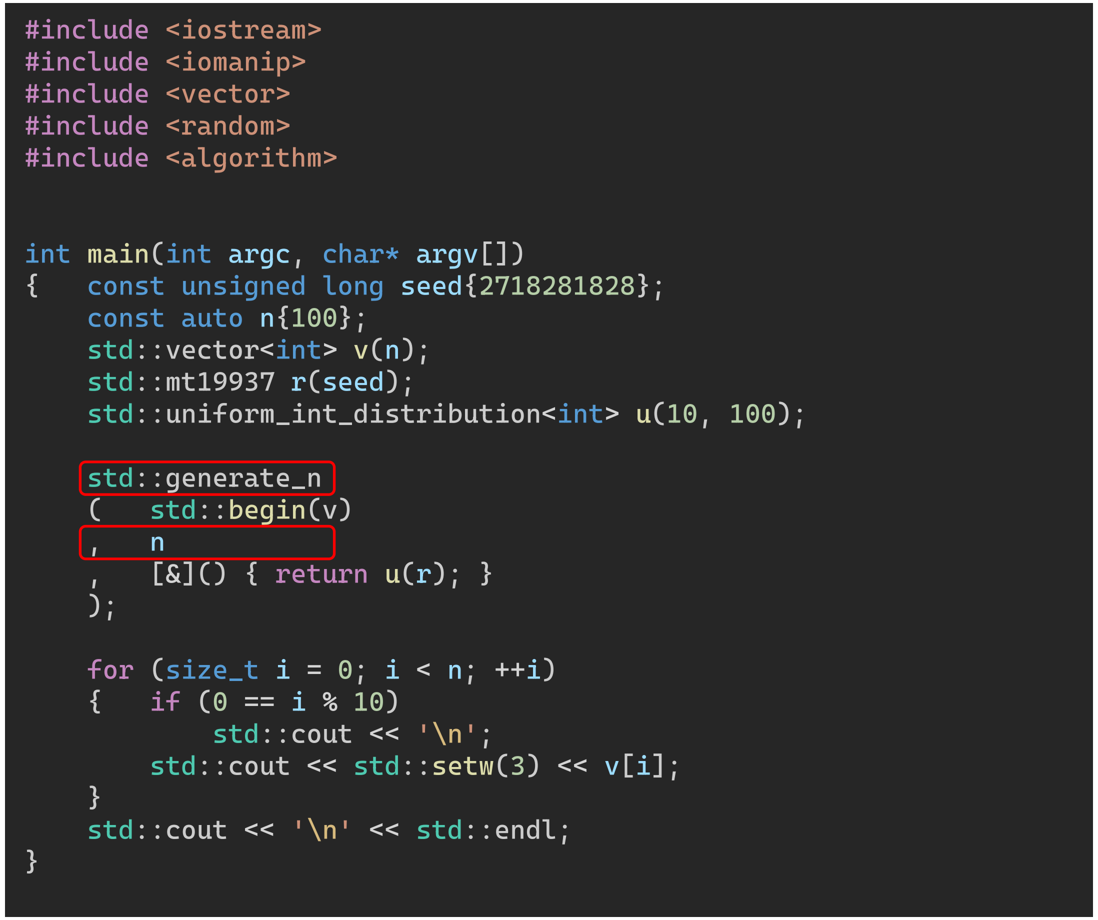


```
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>


int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    std::generate_n
    (   std::begin(v)
    ,   n
    ,   [&]() { return u(r); }
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

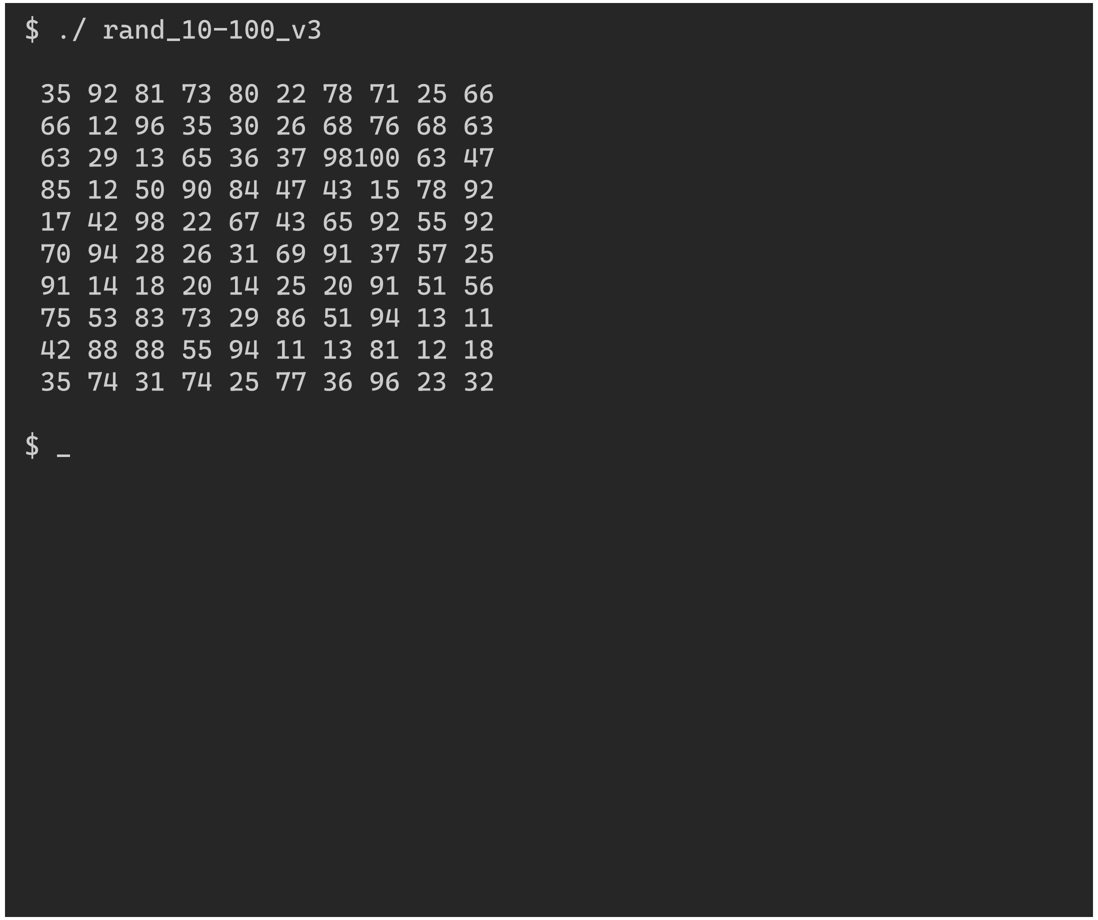


```
$ ./ rand_10-100_v3

 35 92 81 73 80 22 78 71 25 66
 66 12 96 35 30 26 68 76 68 63
 63 29 13 65 36 37 98100 63 47
 85 12 50 90 84 47 43 15 78 92
 17 42 98 22 67 43 65 92 55 92
 70 94 28 26 31 69 91 37 57 25
 91 14 18 20 14 25 20 91 51 56
 75 53 83 73 29 86 51 94 13 11
 42 88 88 55 94 11 13 81 12 18
 35 74 31 74 25 77 36 96 23 32

$ _
```

std:generate_n() with *Lambda*

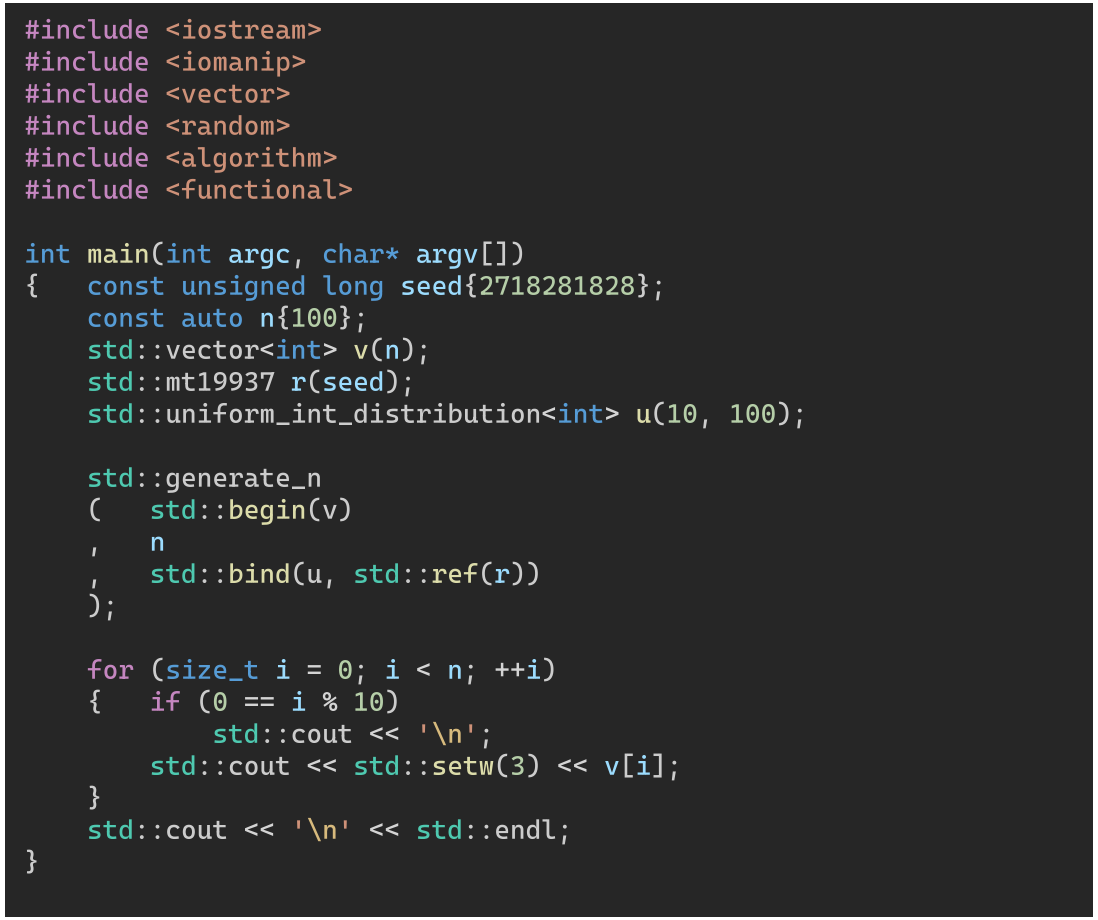


```
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>


int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    std::generate_n
    (   std::begin(v)
    ,   n
    ,   [&]() { return u(r); }
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

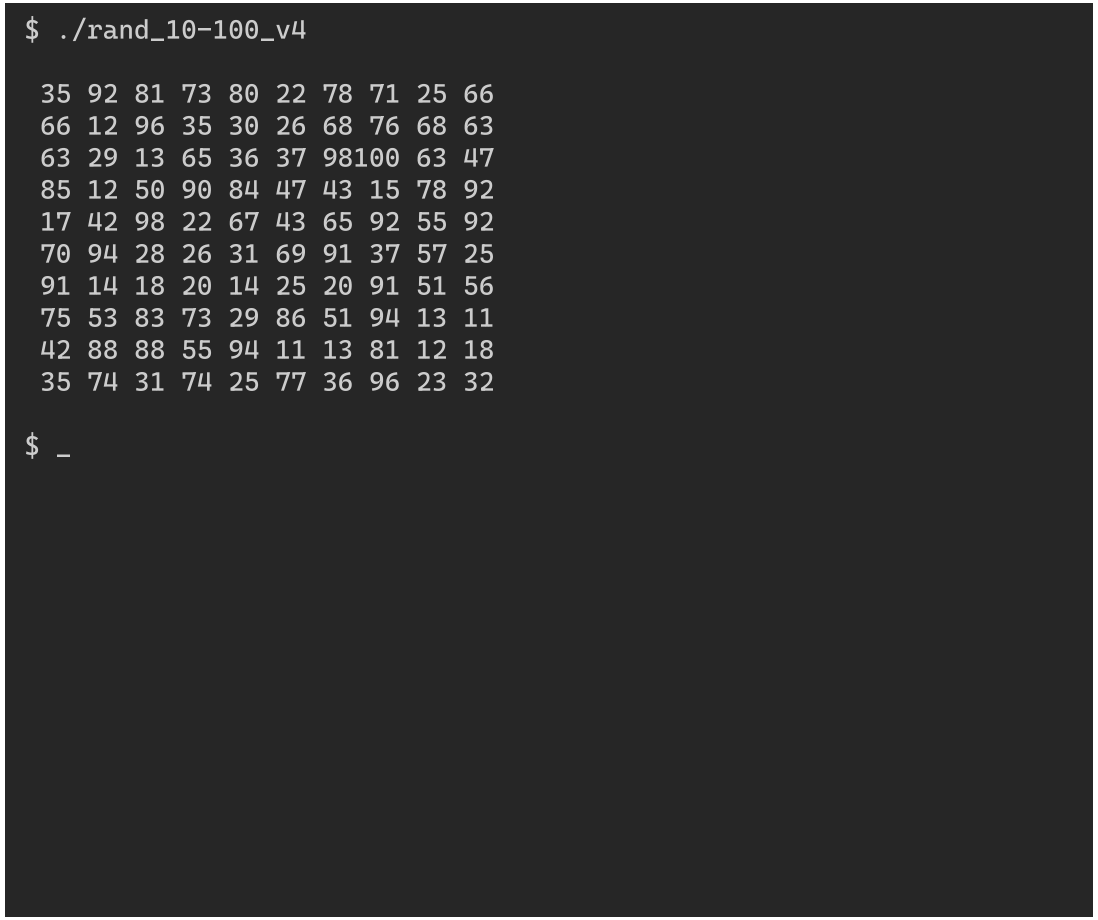


```
$ ./ rand_10-100_v3

 35 92 81 73 80 22 78 71 25 66
 66 12 96 35 30 26 68 76 68 63
 63 29 13 65 36 37 98100 63 47
 85 12 50 90 84 47 43 15 78 92
 17 42 98 22 67 43 65 92 55 92
 70 94 28 26 31 69 91 37 57 25
 91 14 18 20 14 25 20 91 51 56
 75 53 83 73 29 86 51 94 13 11
 42 88 88 55 94 11 13 81 12 18
 35 74 31 74 25 77 36 96 23 32

$ _
```

std:generate_n() with std::bind()

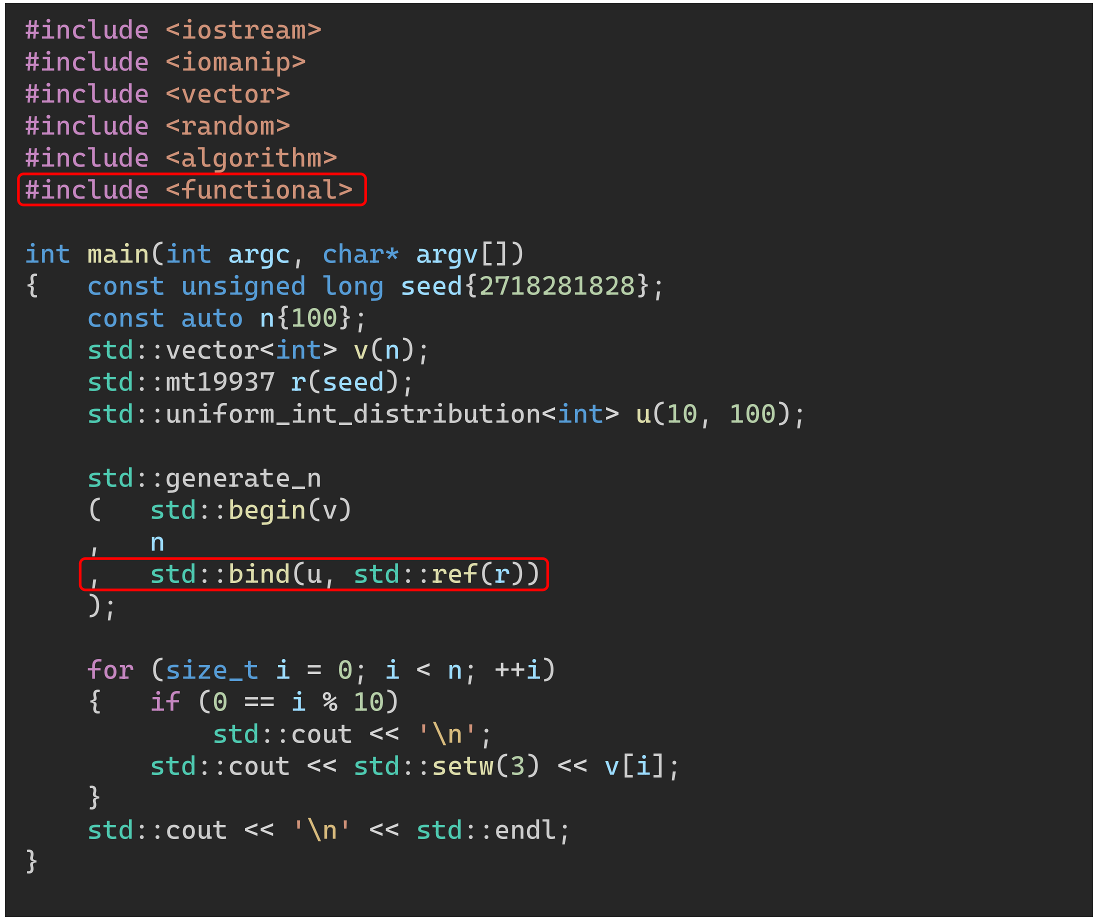


```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>

int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    std::generate_n
    (   std::begin(v)
    ,   n
    ,   std::bind(u, std::ref(r))
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

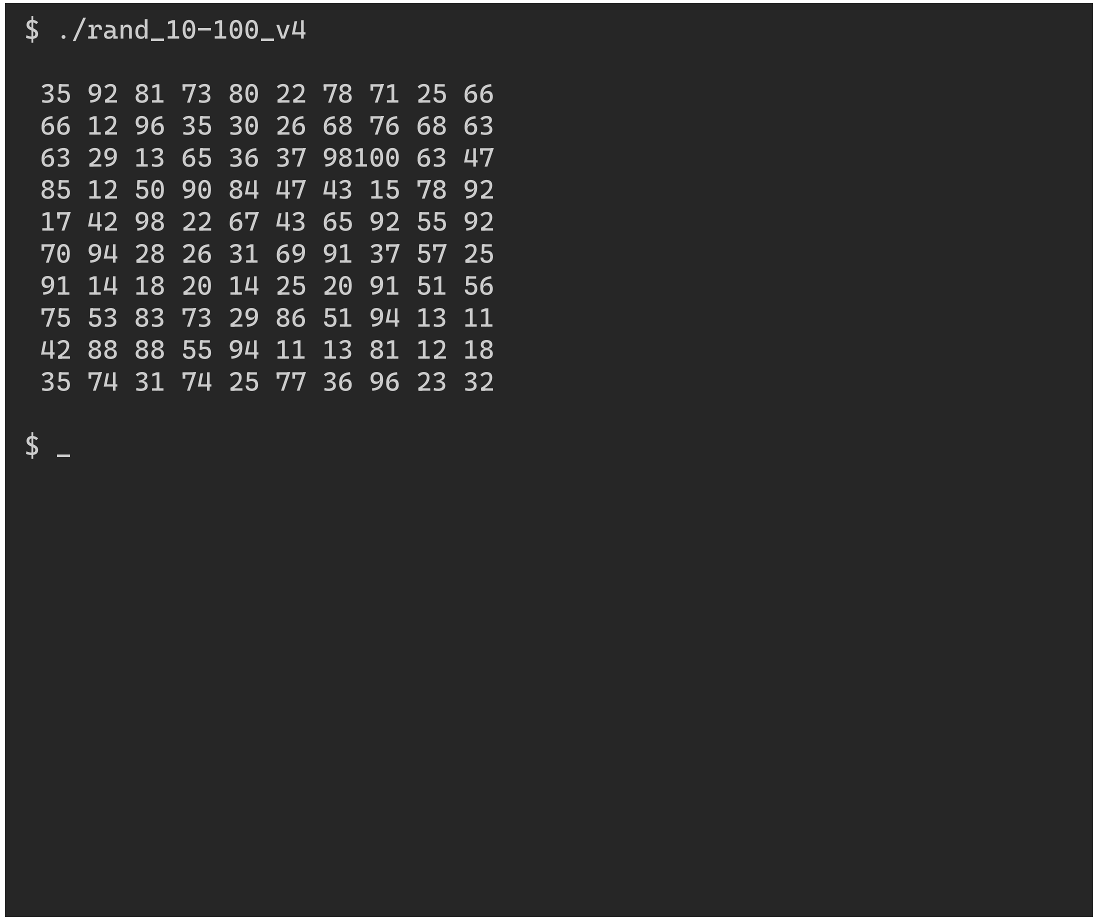


```
$ ./rand_10-100_v4

 35 92 81 73 80 22 78 71 25 66
 66 12 96 35 30 26 68 76 68 63
 63 29 13 65 36 37 98100 63 47
 85 12 50 90 84 47 43 15 78 92
 17 42 98 22 67 43 65 92 55 92
 70 94 28 26 31 69 91 37 57 25
 91 14 18 20 14 25 20 91 51 56
 75 53 83 73 29 86 51 94 13 11
 42 88 88 55 94 11 13 81 12 18
 35 74 31 74 25 77 36 96 23 32

$ _
```

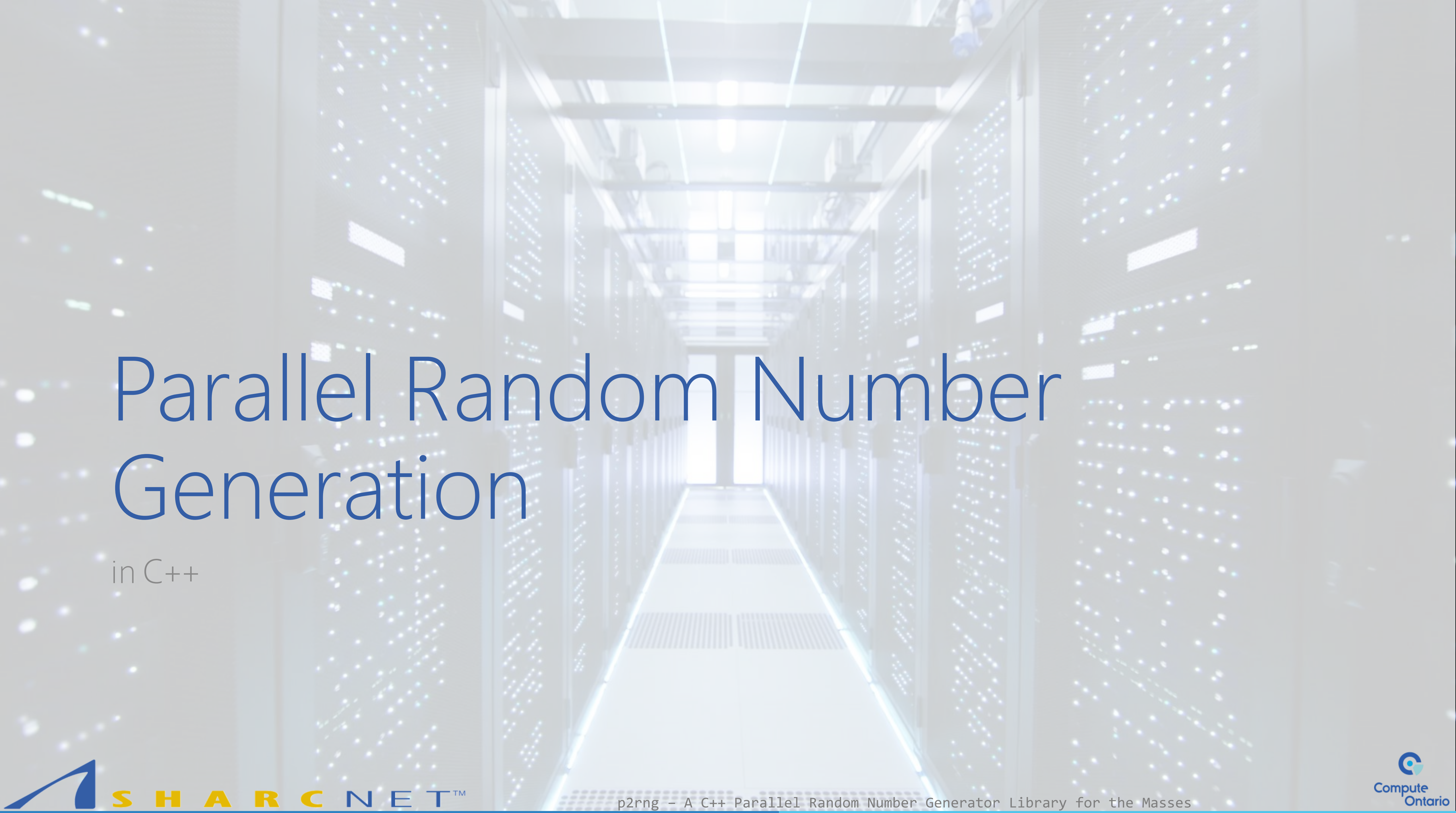
std:generate_n() with std::bind()



```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>

int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    std::generate_n
    (   std::begin(v)
    ,   n
    ,   std::bind(u, std::ref(r))
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```



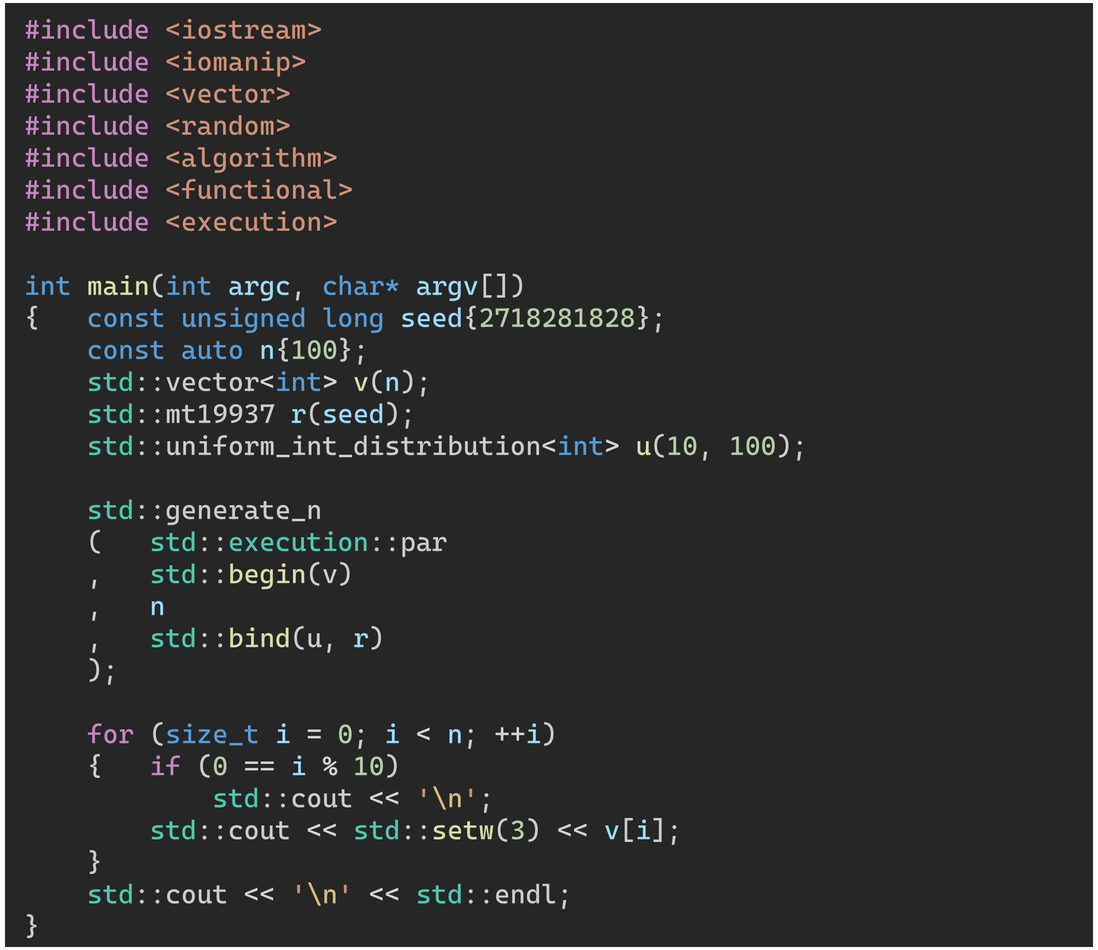
```
$ ./rand_10-100_v4

 35 92 81 73 80 22 78 71 25 66
 66 12 96 35 30 26 68 76 68 63
 63 29 13 65 36 37 98100 63 47
 85 12 50 90 84 47 43 15 78 92
 17 42 98 22 67 43 65 92 55 92
 70 94 28 26 31 69 91 37 57 25
 91 14 18 20 14 25 20 91 51 56
 75 53 83 73 29 86 51 94 13 11
 42 88 88 55 94 11 13 81 12 18
 35 74 31 74 25 77 36 96 23 32

$ _
```



Parallel Random Number Generation

in C++

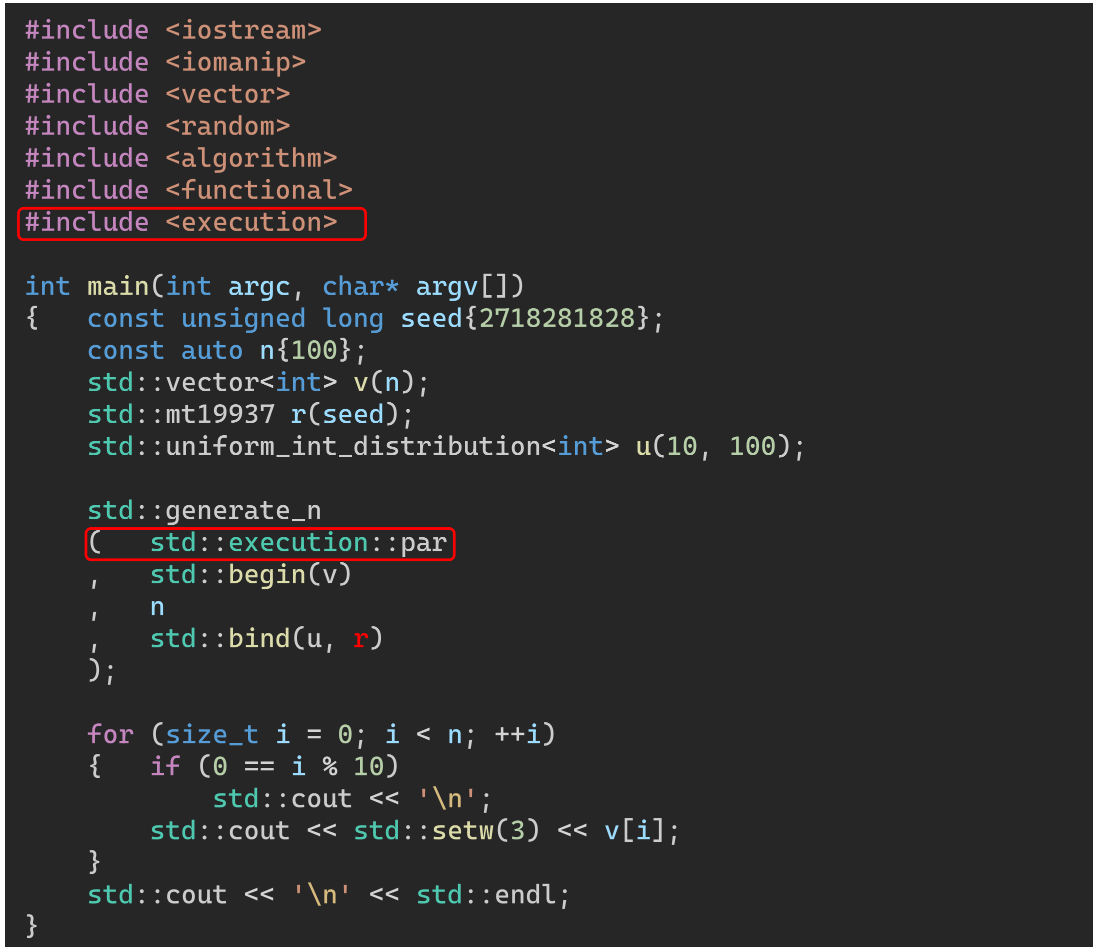
C++17 Parallel `generate_n()` – 1st Attempt



```
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>
#include <execution>

int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    std::generate_n
    (   std::execution::par
    ,   std::begin(v)
    ,   n
    ,   std::bind(u, r)
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

```
$ _
```

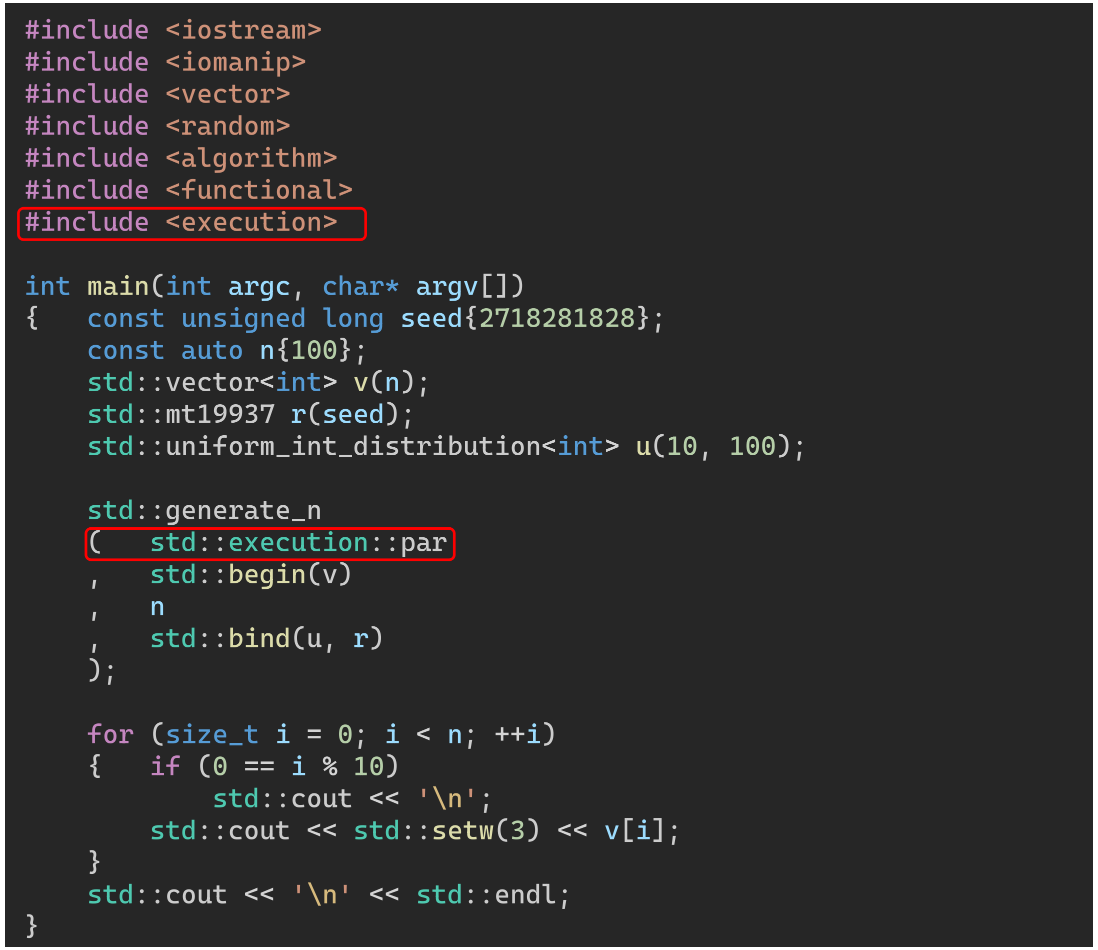
C++17 Parallel `generate_n()` – 1st Attempt



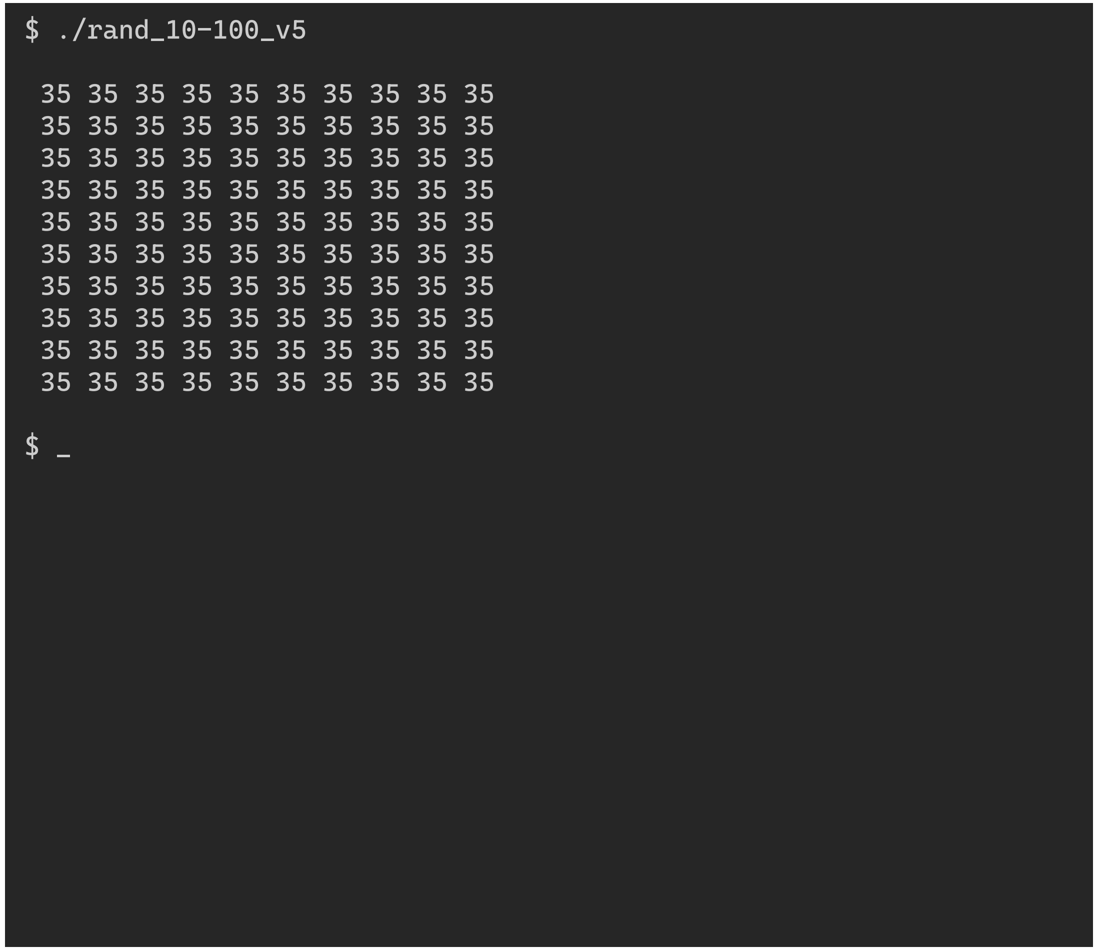
```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>
#include <execution>

int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    std::generate_n
    (   std::execution::par
    ,   std::begin(v)
    ,   n
    ,   std::bind(u, r)
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

```
$ _
```

C++17 Parallel `generate_n()` – 1st Attempt



```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>
#include <execution>

int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    std::generate_n
    (   std::execution::par
    ,   std::begin(v)
    ,   n
    ,   std::bind(u, r)
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```



```
$ ./rand_10-100_v5

 35 35 35 35 35 35 35 35 35 35
 35 35 35 35 35 35 35 35 35 35
 35 35 35 35 35 35 35 35 35 35
 35 35 35 35 35 35 35 35 35 35
 35 35 35 35 35 35 35 35 35 35
 35 35 35 35 35 35 35 35 35 35
 35 35 35 35 35 35 35 35 35 35
 35 35 35 35 35 35 35 35 35 35
 35 35 35 35 35 35 35 35 35 35
 35 35 35 35 35 35 35 35 35 35

$ _
```

C++17 Parallel `generate_n()` – 2nd Attempt

```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>
#include <execution>

int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    std::generate_n
    (   std::execution::par
    ,   std::begin(v)
    ,   n
    ,   std::bind(u, std::ref(r))
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

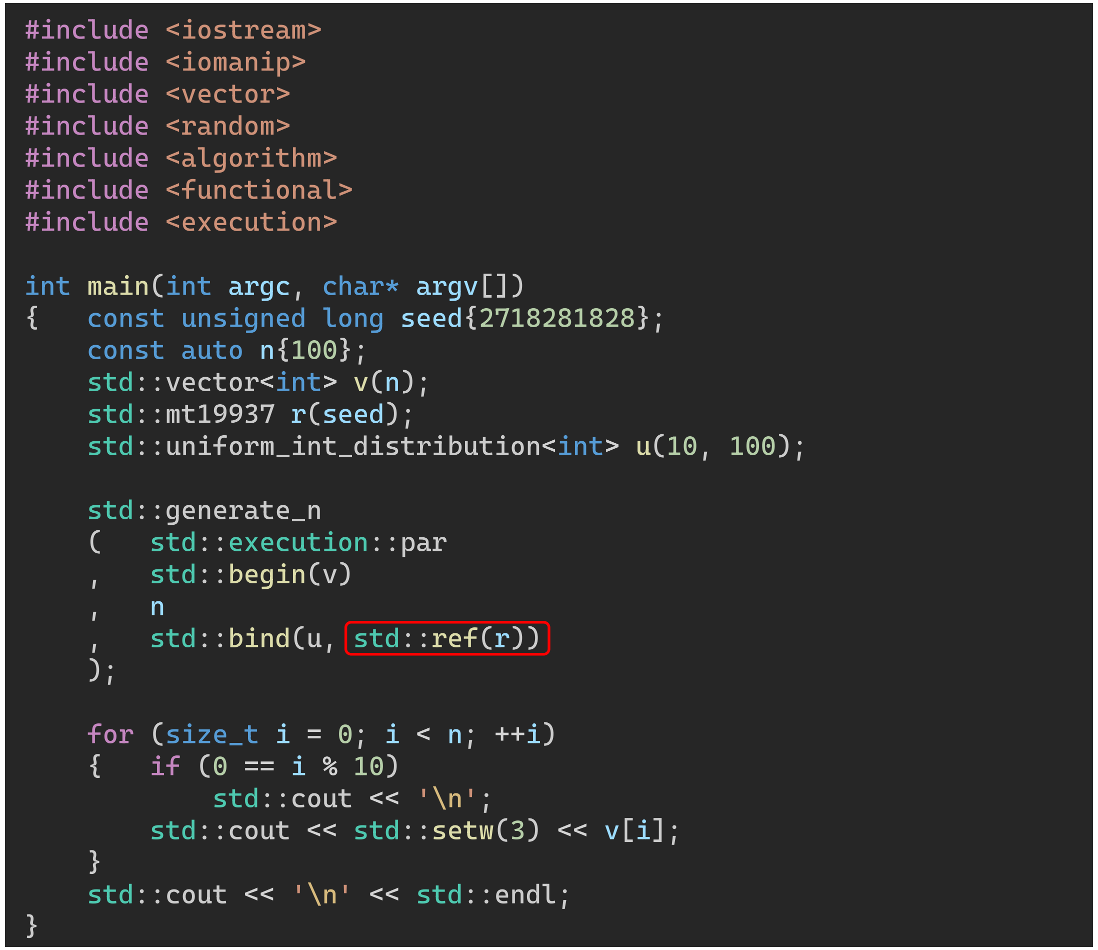
```
$ _
```

C++17 Parallel `generate_n()` – 2nd Attempt

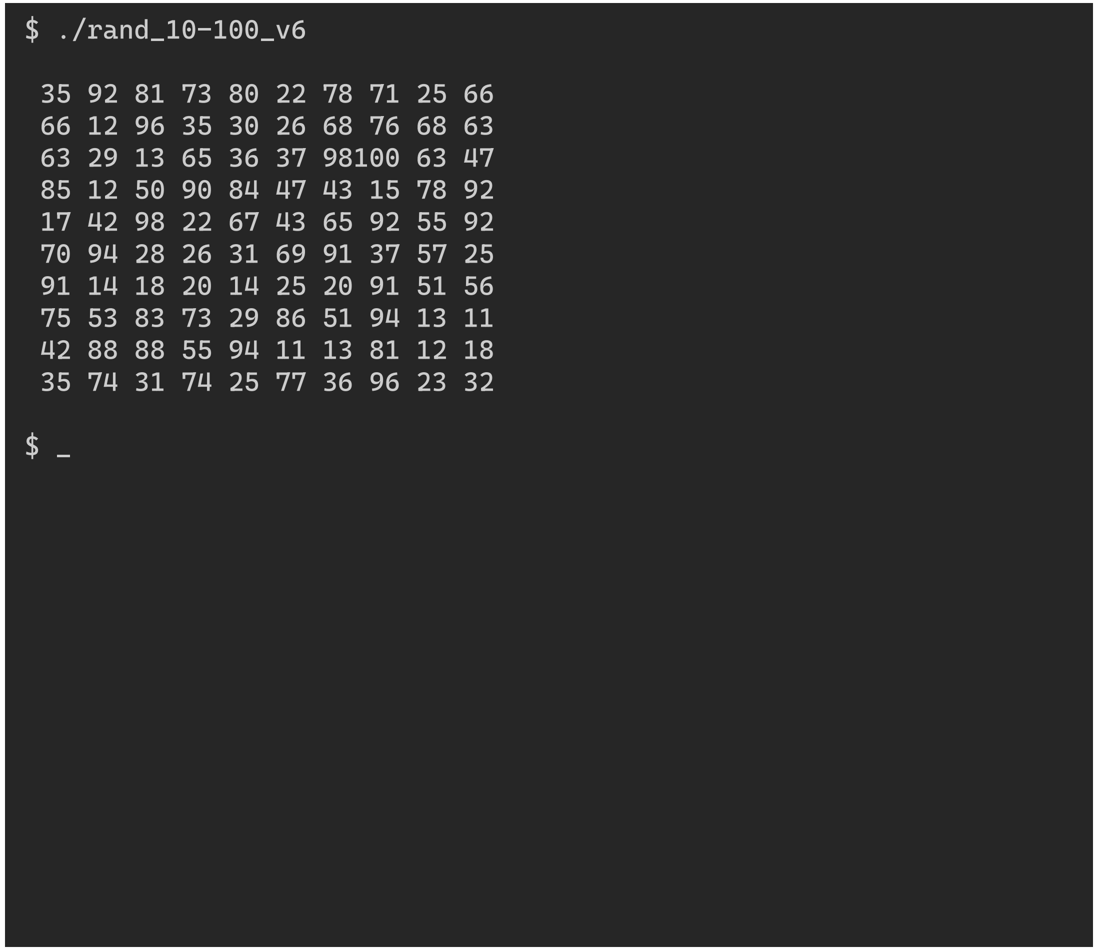
```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>
#include <execution>

int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    std::generate_n
    (   std::execution::par
    ,   std::begin(v)
    ,   n
    ,   std::bind(u, std::ref(r))
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

```
$ _
```

C++17 Parallel `generate_n()` – 2nd Attempt

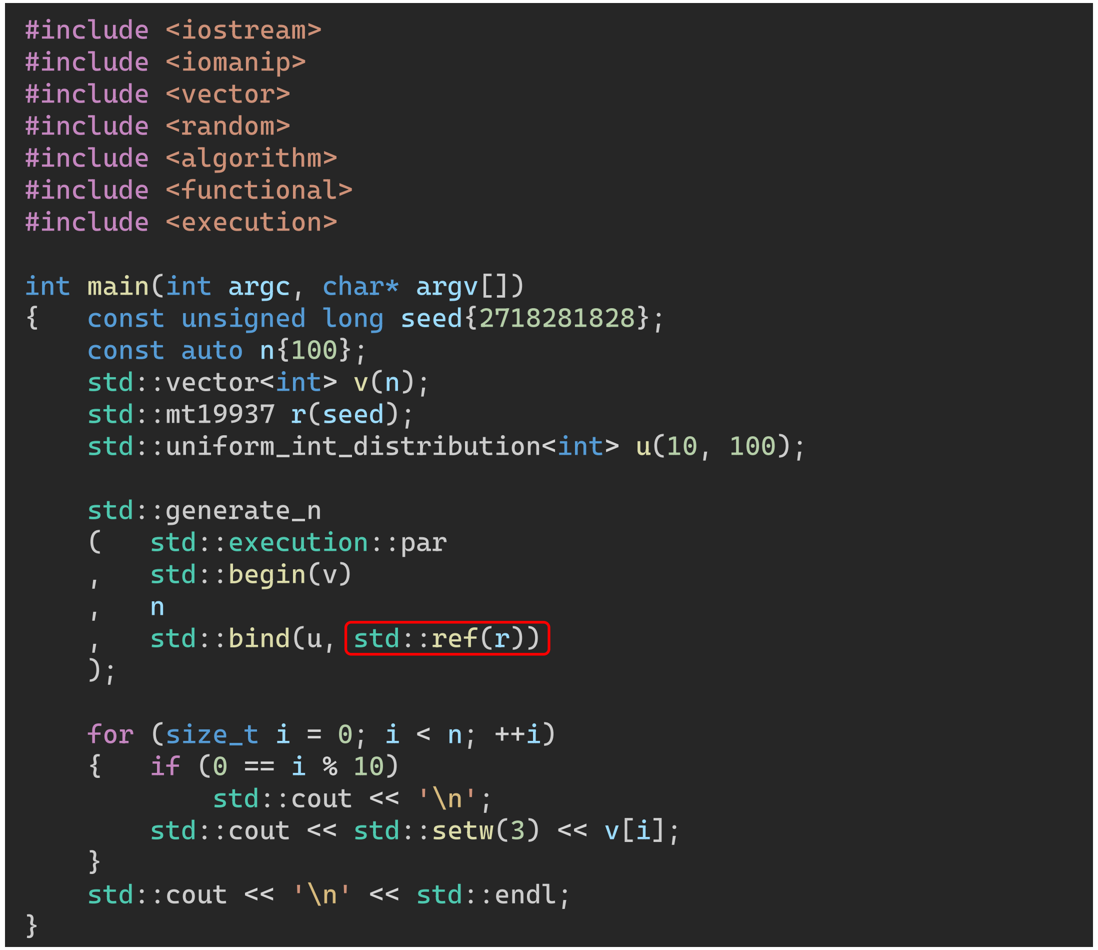


```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>
#include <execution>

int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    std::generate_n
    (   std::execution::par
    ,   std::begin(v)
    ,   n
    ,   std::bind(u, std::ref(r))
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

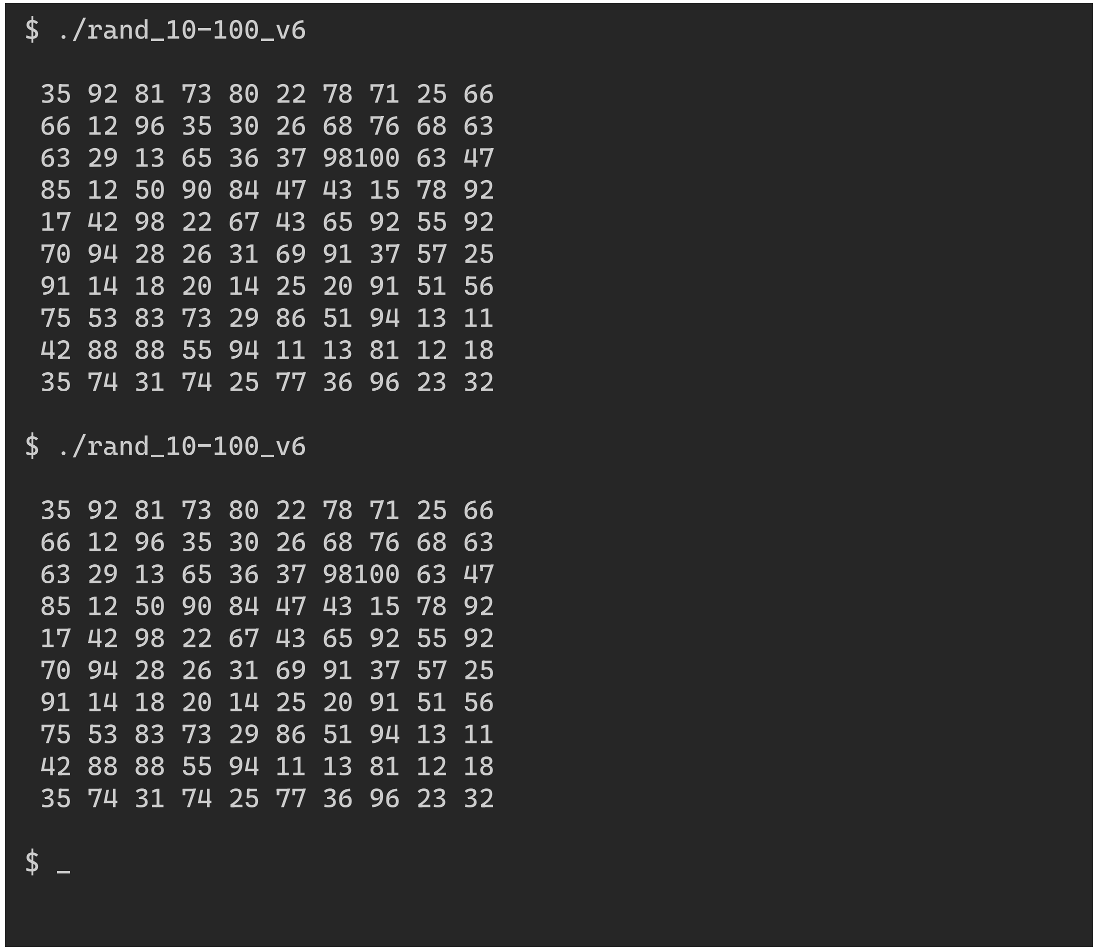


```
$ ./rand_10-100_v6

 35 92 81 73 80 22 78 71 25 66
 66 12 96 35 30 26 68 76 68 63
 63 29 13 65 36 37 98100 63 47
 85 12 50 90 84 47 43 15 78 92
 17 42 98 22 67 43 65 92 55 92
 70 94 28 26 31 69 91 37 57 25
 91 14 18 20 14 25 20 91 51 56
 75 53 83 73 29 86 51 94 13 11
 42 88 88 55 94 11 13 81 12 18
 35 74 31 74 25 77 36 96 23 32

$ _
```

C++17 Parallel `generate_n()` – 2nd Attempt



```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>
#include <execution>

int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 100);

    std::generate_n
    (   std::execution::par
    ,   std::begin(v)
    ,   n
    ,   std::bind(u, std::ref(r))
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

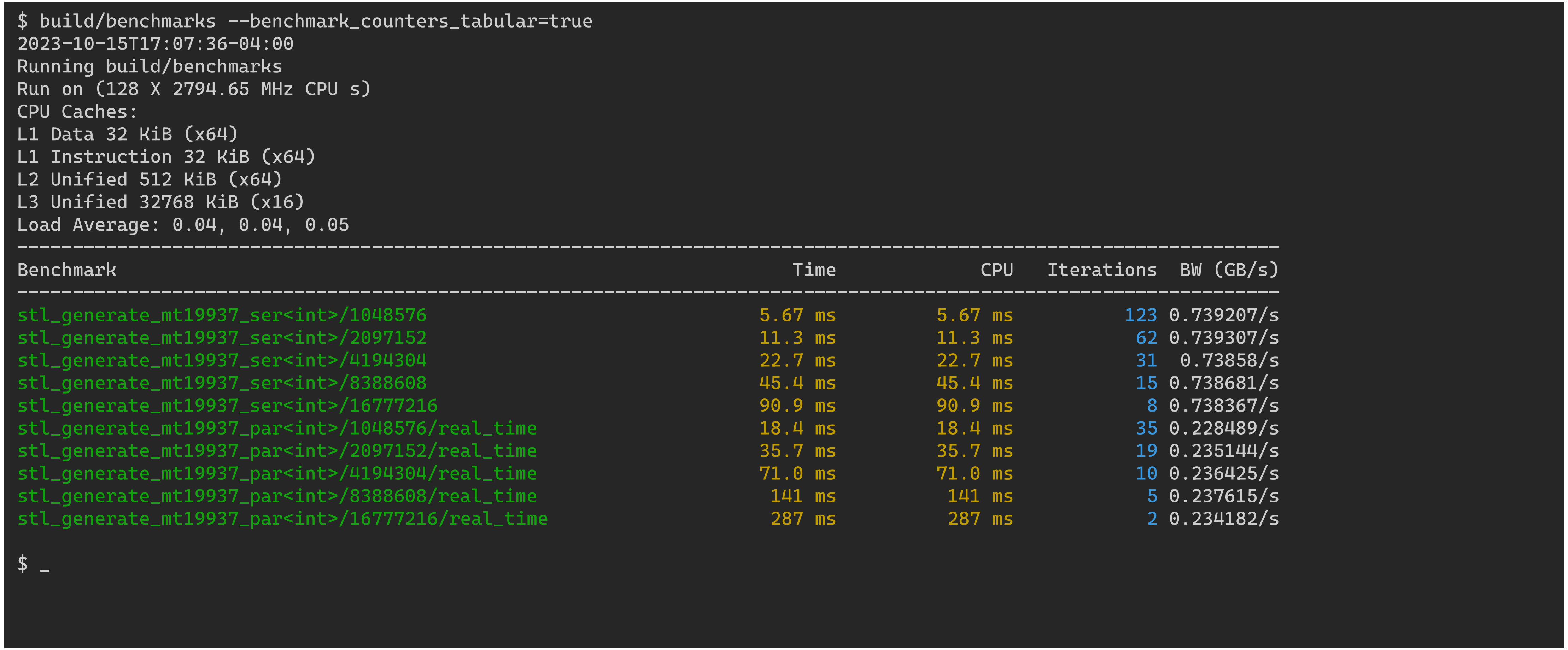


```
$ ./rand_10-100_v6

 35 92 81 73 80 22 78 71 25 66
 66 12 96 35 30 26 68 76 68 63
 63 29 13 65 36 37 98100 63 47
 85 12 50 90 84 47 43 15 78 92
 17 42 98 22 67 43 65 92 55 92
 70 94 28 26 31 69 91 37 57 25
 91 14 18 20 14 25 20 91 51 56
 75 53 83 73 29 86 51 94 13 11
 42 88 88 55 94 11 13 81 12 18
 35 74 31 74 25 77 36 96 23 32

$ ./rand_10-100_v6

 35 92 81 73 80 22 78 71 25 66
 66 12 96 35 30 26 68 76 68 63
 63 29 13 65 36 37 98100 63 47
 85 12 50 90 84 47 43 15 78 92
 17 42 98 22 67 43 65 92 55 92
 70 94 28 26 31 69 91 37 57 25
 91 14 18 20 14 25 20 91 51 56
 75 53 83 73 29 86 51 94 13 11
 42 88 88 55 94 11 13 81 12 18
 35 74 31 74 25 77 36 96 23 32

$ _
```

Using bash’s `time` Command for timing

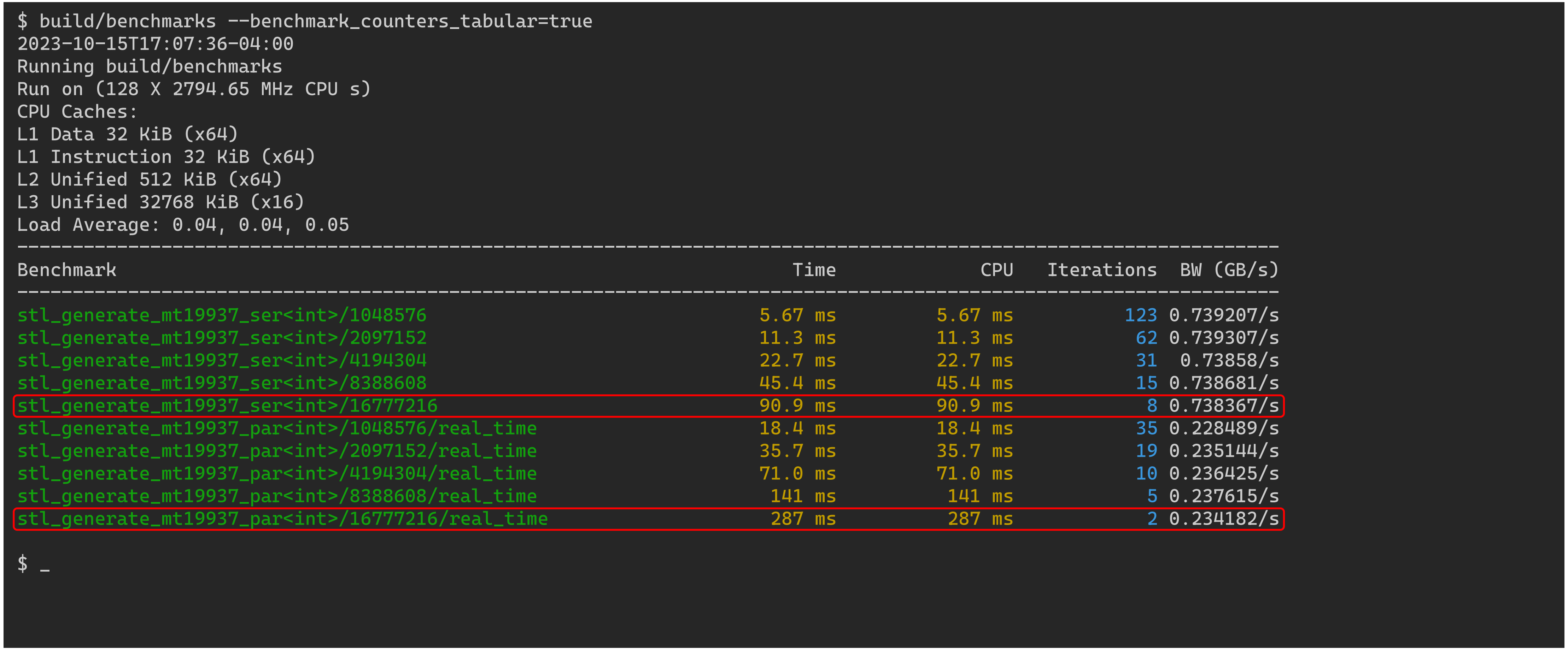
```
$ time build/rand_10-100_v4 # serial version

real 0m0.765s
user 0m0.745s
sys 0m0.020s

$ time build/rand_10-100_v6 # C++17 parallel version

real 0m1.761s
user 3m35.483s
sys 0m0.522s
```

Benchmark Results on AMD Epic 7543 CPU



```
$ build/benchmarks --benchmark_counters_tabular=true
2023-10-15T17:07:36-04:00
Running build/benchmarks
Run on (128 X 2794.65 MHz CPU s)
CPU Caches:
L1 Data 32 KiB (x64)
L1 Instruction 32 KiB (x64)
L2 Unified 512 KiB (x64)
L3 Unified 32768 KiB (x16)
Load Average: 0.04, 0.04, 0.05
------------------------------------------------------------------------------------------------------------------
Benchmark                                                             Time             CPU   Iterations  BW (GB/s)
------------------------------------------------------------------------------------------------------------------
stl_generate_mt19937_ser<int>/1048576                              5.67 ms         5.67 ms          123 0.739207/s
stl_generate_mt19937_ser<int>/2097152                              11.3 ms         11.3 ms           62 0.739307/s
stl_generate_mt19937_ser<int>/4194304                              22.7 ms         22.7 ms           31  0.73858/s
stl_generate_mt19937_ser<int>/8388608                              45.4 ms         45.4 ms           15 0.738681/s
stl_generate_mt19937_ser<int>/16777216                             90.9 ms         90.9 ms            8 0.738367/s
stl_generate_mt19937_par<int>/1048576/real_time                    18.4 ms         18.4 ms           35 0.228489/s
stl_generate_mt19937_par<int>/2097152/real_time                    35.7 ms         35.7 ms           19 0.235144/s
stl_generate_mt19937_par<int>/4194304/real_time                    71.0 ms         71.0 ms           10 0.236425/s
stl_generate_mt19937_par<int>/8388608/real_time                     141 ms          141 ms            5 0.237615/s
stl_generate_mt19937_par<int>/16777216/real_time                    287 ms          287 ms            2 0.234182/s

$ _
```

Benchmark Results on AMD Epic 7543 CPU



```
$ build/benchmarks --benchmark_counters_tabular=true
2023-10-15T17:07:36-04:00
Running build/benchmarks
Run on (128 X 2794.65 MHz CPU s)
CPU Caches:
L1 Data 32 KiB (x64)
L1 Instruction 32 KiB (x64)
L2 Unified 512 KiB (x64)
L3 Unified 32768 KiB (x16)
Load Average: 0.04, 0.04, 0.05
------------------------------------------------------------------------------------------------------------------
Benchmark                                                             Time             CPU   Iterations  BW (GB/s)
------------------------------------------------------------------------------------------------------------------
stl_generate_mt19937_ser<int>/1048576                              5.67 ms         5.67 ms          123 0.739207/s
stl_generate_mt19937_ser<int>/2097152                              11.3 ms         11.3 ms           62 0.739307/s
stl_generate_mt19937_ser<int>/4194304                              22.7 ms         22.7 ms           31  0.73858/s
stl_generate_mt19937_ser<int>/8388608                              45.4 ms         45.4 ms           15 0.738681/s
stl_generate_mt19937_ser<int>/16777216                             90.9 ms         90.9 ms            8 0.738367/s
stl_generate_mt19937_par<int>/1048576/real_time                    18.4 ms         18.4 ms           35 0.228489/s
stl_generate_mt19937_par<int>/2097152/real_time                    35.7 ms         35.7 ms           19 0.235144/s
stl_generate_mt19937_par<int>/4194304/real_time                    71.0 ms         71.0 ms           10 0.236425/s
stl_generate_mt19937_par<int>/8388608/real_time                     141 ms          141 ms            5 0.237615/s
stl_generate_mt19937_par<int>/16777216/real_time                    287 ms          287 ms            2 0.234182/s

$ _
```

Verdict

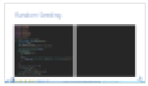
Pros

- Easy to do

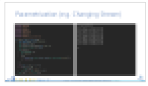
Cons

- Terrible performance
- Possible correlations in subsequences
- Cannot play fair

General Parallelization Techniques



Random Seeding

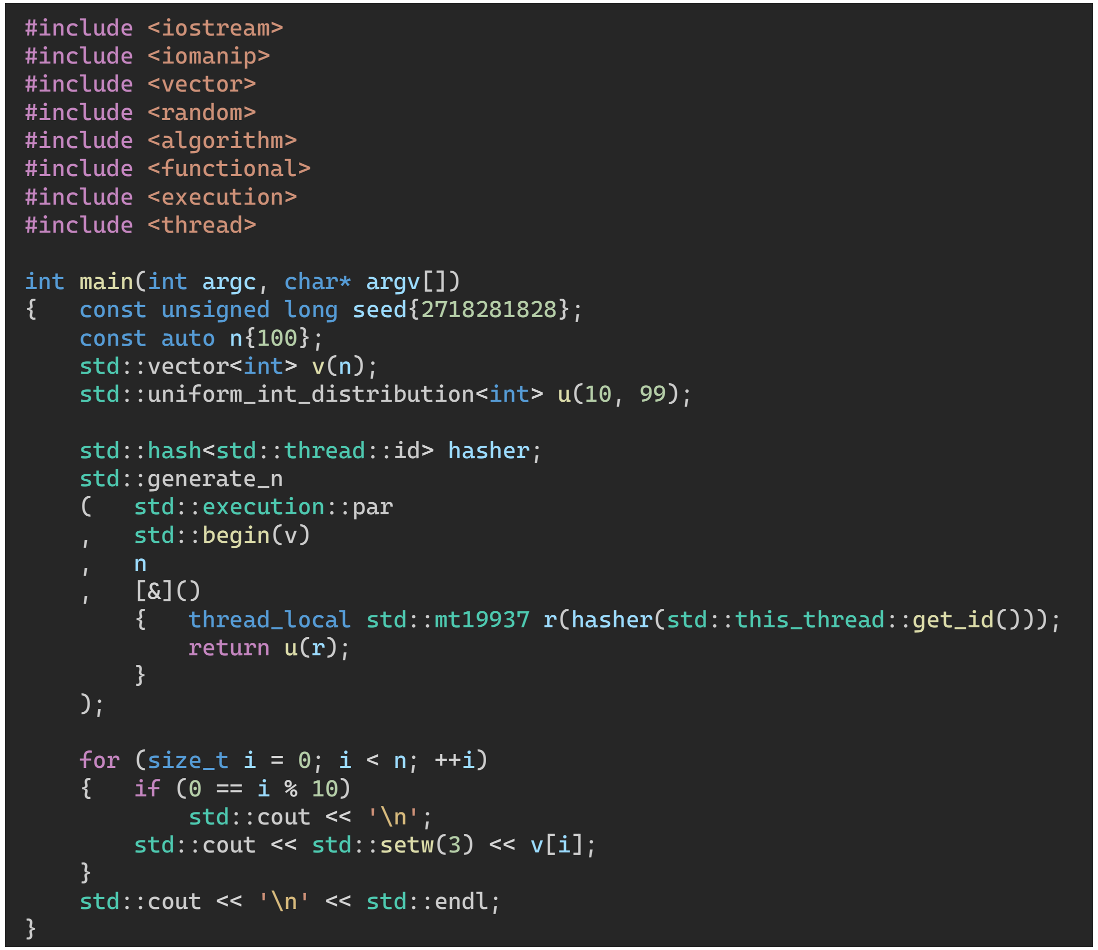


Parametrization

Leapfrog

Block Splitting

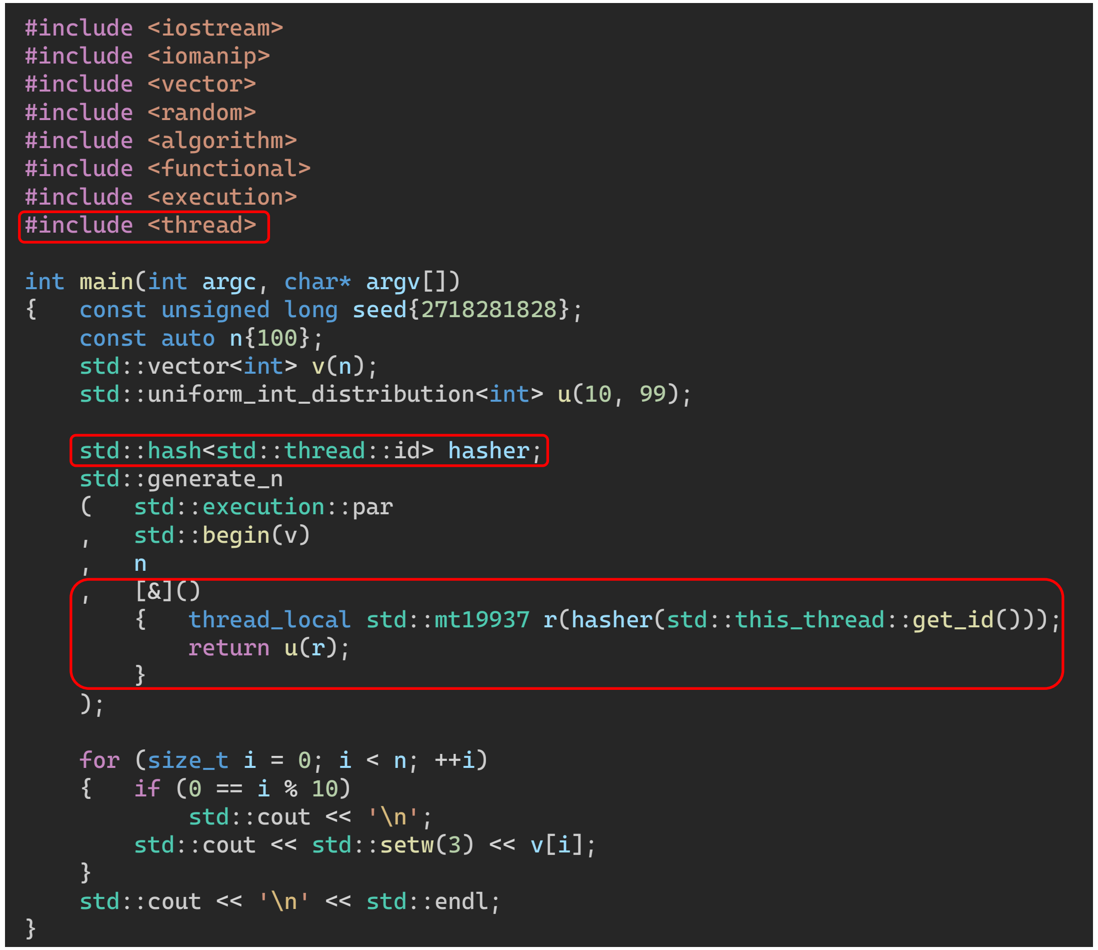
Random Seeding



```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>
#include <execution>
#include <thread>

int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::uniform_int_distribution<int> u(10, 99);

    std::hash<std::thread::id> hasher;
    std::generate_n
    (   std::execution::par
    ,   std::begin(v)
    ,   n
    ,   [&]()
        {   thread_local std::mt19937 r(hasher(std::this_thread::get_id()));
            return u(r);
        }
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

```
$ _
```

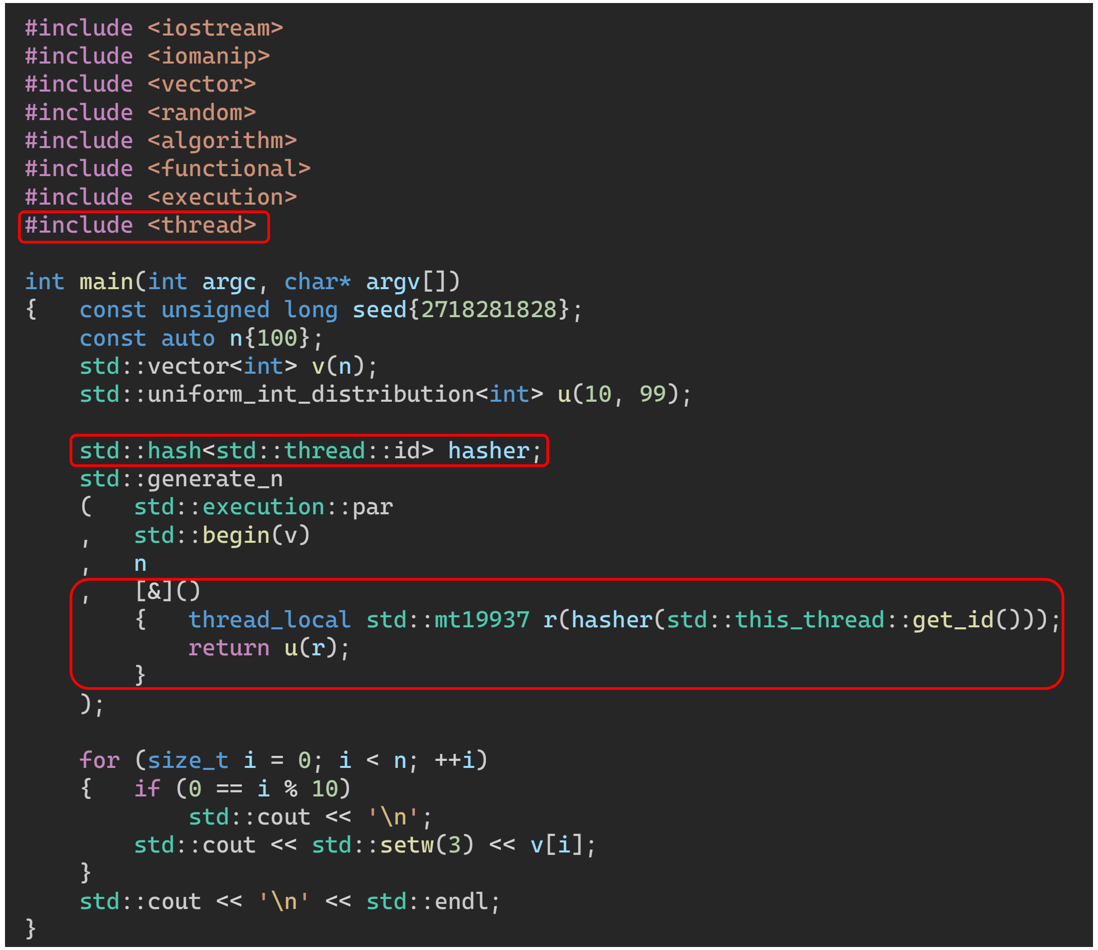
Random Seeding



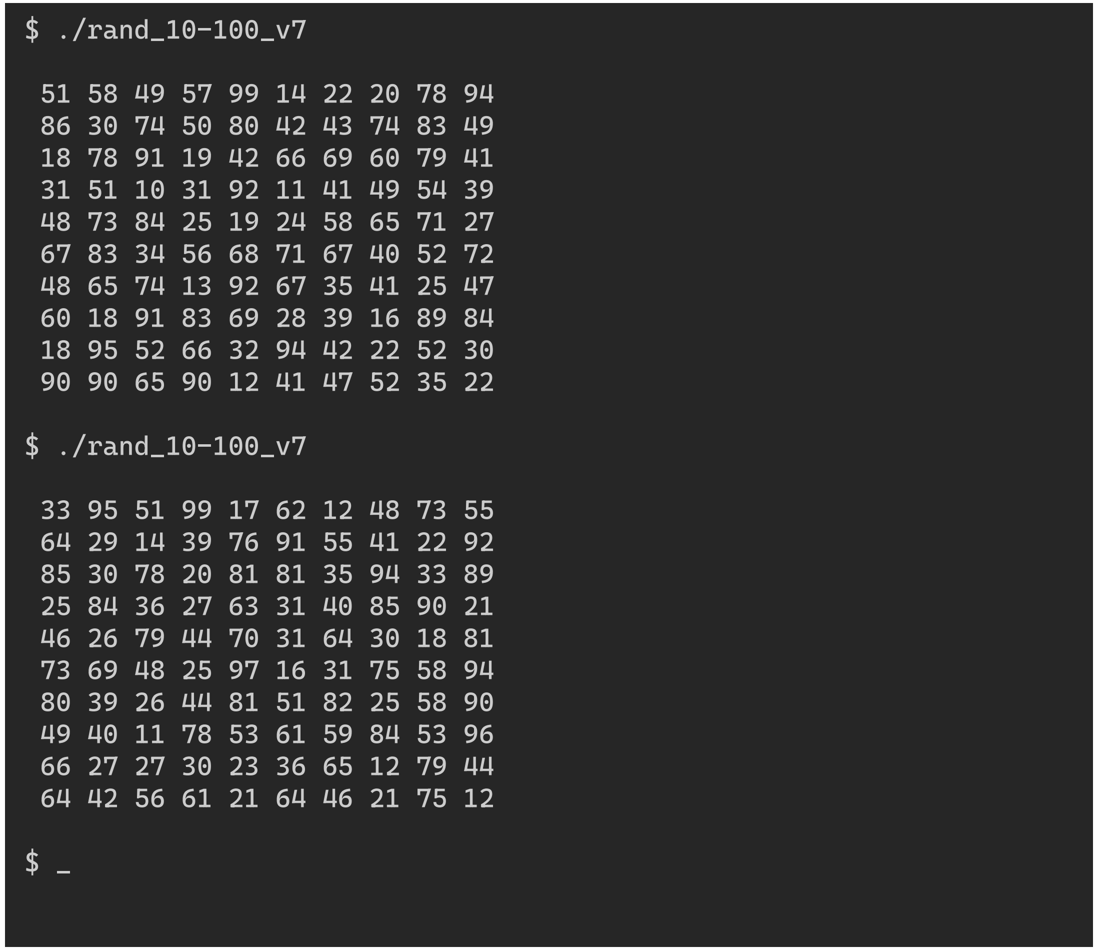
```
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>
#include <execution>
#include <thread>

int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::uniform_int_distribution<int> u(10, 99);

    std::hash<std::thread::id> hasher;
    std::generate_n
    (   std::execution::par
    ,   std::begin(v)
    ,   n
    ,   [&]()
        {   thread_local std::mt19937 r(hasher(std::this_thread::get_id()));
            return u(r);
        }
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

```
$ _
```

Random Seeding

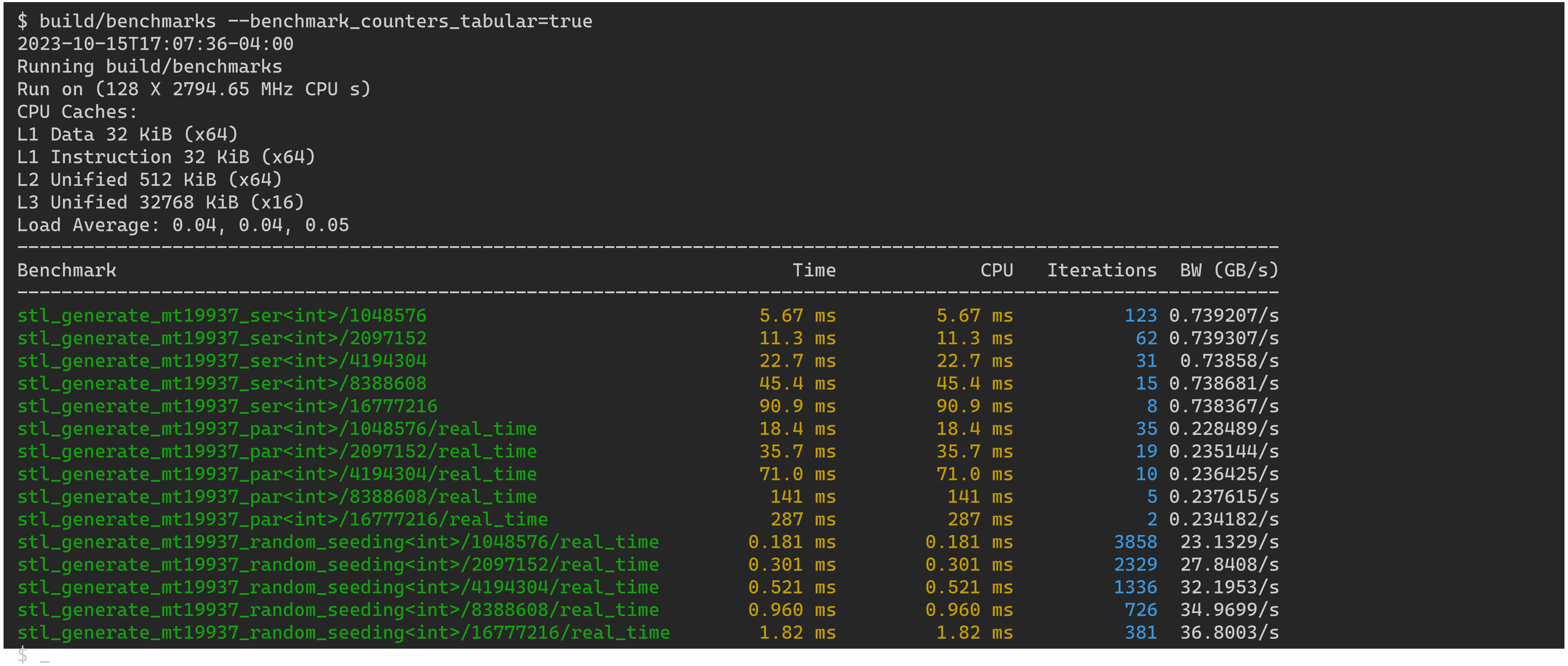


```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>
#include <execution>
#include <thread>

int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::uniform_int_distribution<int> u(10, 99);

    std::hash<std::thread::id> hasher;
    std::generate_n
    (   std::execution::par
    ,   std::begin(v)
    ,   n
    ,   [&]()
        {   thread_local std::mt19937 r(hasher(std::this_thread::get_id()));
            return u(r);
        }
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```



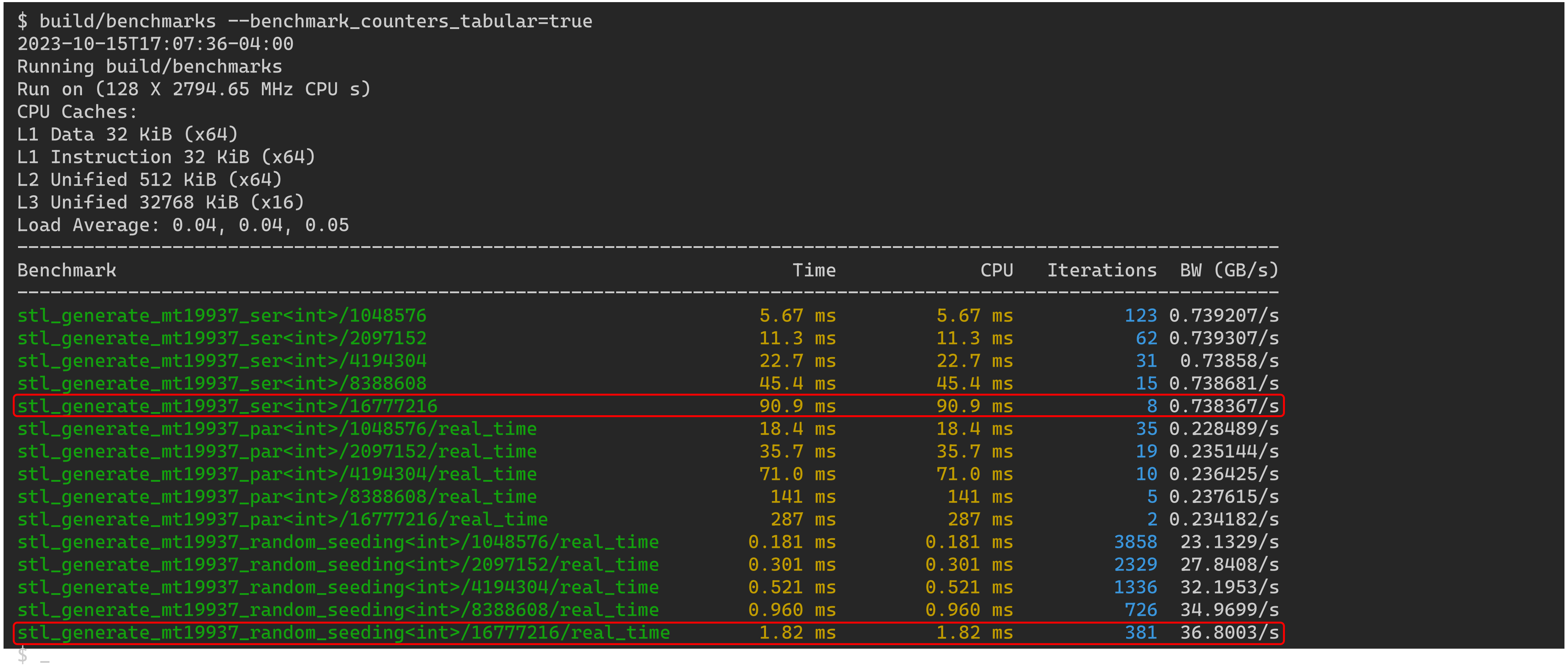
```
$ ./rand_10-100_v7

 51 58 49 57 99 14 22 20 78 94
 86 30 74 50 80 42 43 74 83 49
 18 78 91 19 42 66 69 60 79 41
 31 51 10 31 92 11 41 49 54 39
 48 73 84 25 19 24 58 65 71 27
 67 83 34 56 68 71 67 40 52 72
 48 65 74 13 92 67 35 41 25 47
 60 18 91 83 69 28 39 16 89 84
 18 95 52 66 32 94 42 22 52 30
 90 90 65 90 12 41 47 52 35 22

$ ./rand_10-100_v7

 33 95 51 99 17 62 12 48 73 55
 64 29 14 39 76 91 55 41 22 92
 85 30 78 20 81 81 35 94 33 89
 25 84 36 27 63 31 40 85 90 21
 46 26 79 44 70 31 64 30 18 81
 73 69 48 25 97 16 31 75 58 94
 80 39 26 44 81 51 82 25 58 90
 49 40 11 78 53 61 59 84 53 96
 66 27 27 30 23 36 65 12 79 44
 64 42 56 61 21 64 46 21 75 12

$ _
```

Random Seeding – Benchmarks (AMD Epic 7543 CPU)



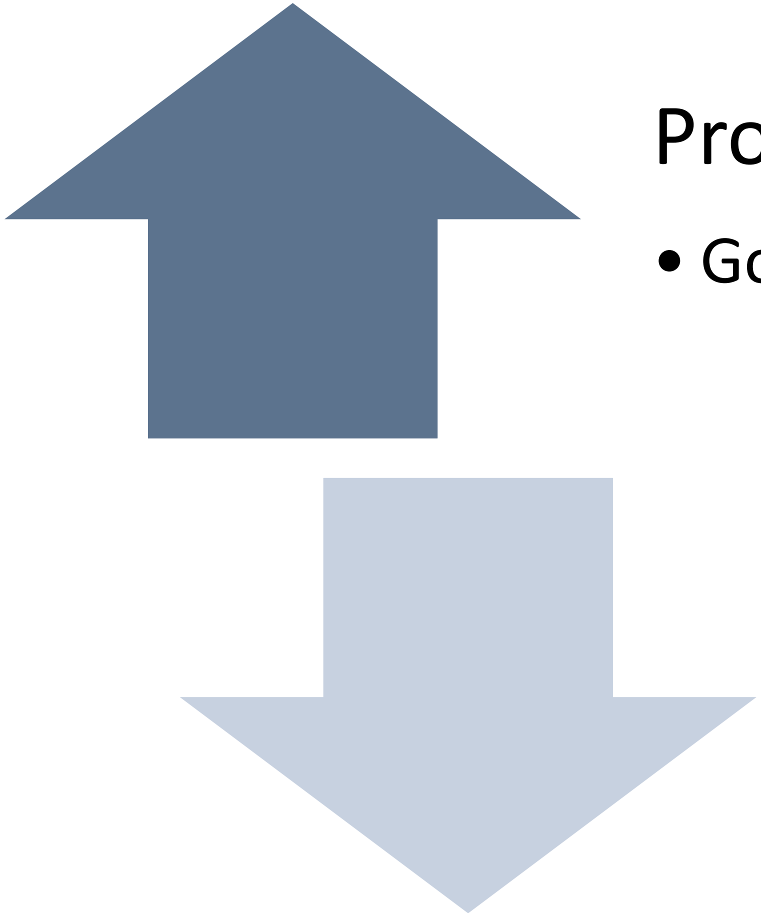
```
$ build/benchmarks --benchmark_counters_tabular=true
2023-10-15T17:07:36-04:00
Running build/benchmarks
Run on (128 X 2794.65 MHz CPU s)
CPU Caches:
L1 Data 32 KiB (x64)
L1 Instruction 32 KiB (x64)
L2 Unified 512 KiB (x64)
L3 Unified 32768 KiB (x16)
Load Average: 0.04, 0.04, 0.05
------------------------------------------------------------------------------------------------------------------
Benchmark                                                             Time             CPU   Iterations  BW (GB/s)
------------------------------------------------------------------------------------------------------------------
stl_generate_mt19937_ser<int>/1048576                              5.67 ms         5.67 ms          123 0.739207/s
stl_generate_mt19937_ser<int>/2097152                              11.3 ms         11.3 ms           62 0.739307/s
stl_generate_mt19937_ser<int>/4194304                              22.7 ms         22.7 ms           31  0.73858/s
stl_generate_mt19937_ser<int>/8388608                              45.4 ms         45.4 ms           15 0.738681/s
stl_generate_mt19937_ser<int>/16777216                             90.9 ms         90.9 ms            8 0.738367/s
stl_generate_mt19937_par<int>/1048576/real_time                    18.4 ms         18.4 ms           35 0.228489/s
stl_generate_mt19937_par<int>/2097152/real_time                    35.7 ms         35.7 ms           19 0.235144/s
stl_generate_mt19937_par<int>/4194304/real_time                    71.0 ms         71.0 ms           10 0.236425/s
stl_generate_mt19937_par<int>/8388608/real_time                     141 ms          141 ms            5 0.237615/s
stl_generate_mt19937_par<int>/16777216/real_time                    287 ms          287 ms            2 0.234182/s
stl_generate_mt19937_random_seeding<int>/1048576/real_time        0.181 ms        0.181 ms         3858  23.1329/s
stl_generate_mt19937_random_seeding<int>/2097152/real_time        0.301 ms        0.301 ms         2329  27.8408/s
stl_generate_mt19937_random_seeding<int>/4194304/real_time        0.521 ms        0.521 ms         1336  32.1953/s
stl_generate_mt19937_random_seeding<int>/8388608/real_time        0.960 ms        0.960 ms          726  34.9699/s
stl_generate_mt19937_random_seeding<int>/16777216/real_time        1.82 ms         1.82 ms          381  36.8003/s
$ _
```

Random Seeding – Benchmarks (AMD Epic 7543 CPU)



```
$ build/benchmarks --benchmark_counters_tabular=true
2023-10-15T17:07:36-04:00
Running build/benchmarks
Run on (128 X 2794.65 MHz CPU s)
CPU Caches:
L1 Data 32 KiB (x64)
L1 Instruction 32 KiB (x64)
L2 Unified 512 KiB (x64)
L3 Unified 32768 KiB (x16)
Load Average: 0.04, 0.04, 0.05
------------------------------------------------------------------------------------------------------------------
Benchmark                                                             Time             CPU   Iterations  BW (GB/s)
------------------------------------------------------------------------------------------------------------------
stl_generate_mt19937_ser<int>/1048576                              5.67 ms         5.67 ms          123 0.739207/s
stl_generate_mt19937_ser<int>/2097152                              11.3 ms         11.3 ms           62 0.739307/s
stl_generate_mt19937_ser<int>/4194304                              22.7 ms         22.7 ms           31  0.73858/s
stl_generate_mt19937_ser<int>/8388608                              45.4 ms         45.4 ms           15 0.738681/s
stl_generate_mt19937_ser<int>/16777216                             90.9 ms         90.9 ms            8 0.738367/s
stl_generate_mt19937_par<int>/1048576/real_time                    18.4 ms         18.4 ms           35 0.228489/s
stl_generate_mt19937_par<int>/2097152/real_time                    35.7 ms         35.7 ms           19 0.235144/s
stl_generate_mt19937_par<int>/4194304/real_time                    71.0 ms         71.0 ms           10 0.236425/s
stl_generate_mt19937_par<int>/8388608/real_time                     141 ms          141 ms            5 0.237615/s
stl_generate_mt19937_par<int>/16777216/real_time                    287 ms          287 ms            2 0.234182/s
stl_generate_mt19937_random_seeding<int>/1048576/real_time        0.181 ms        0.181 ms         3858  23.1329/s
stl_generate_mt19937_random_seeding<int>/2097152/real_time        0.301 ms        0.301 ms         2329  27.8408/s
stl_generate_mt19937_random_seeding<int>/4194304/real_time        0.521 ms        0.521 ms         1336  32.1953/s
stl_generate_mt19937_random_seeding<int>/8388608/real_time        0.960 ms        0.960 ms          726  34.9699/s
stl_generate_mt19937_random_seeding<int>/16777216/real_time        1.82 ms         1.82 ms          381  36.8003/s
$ _
```

Random Seeding – Verdict



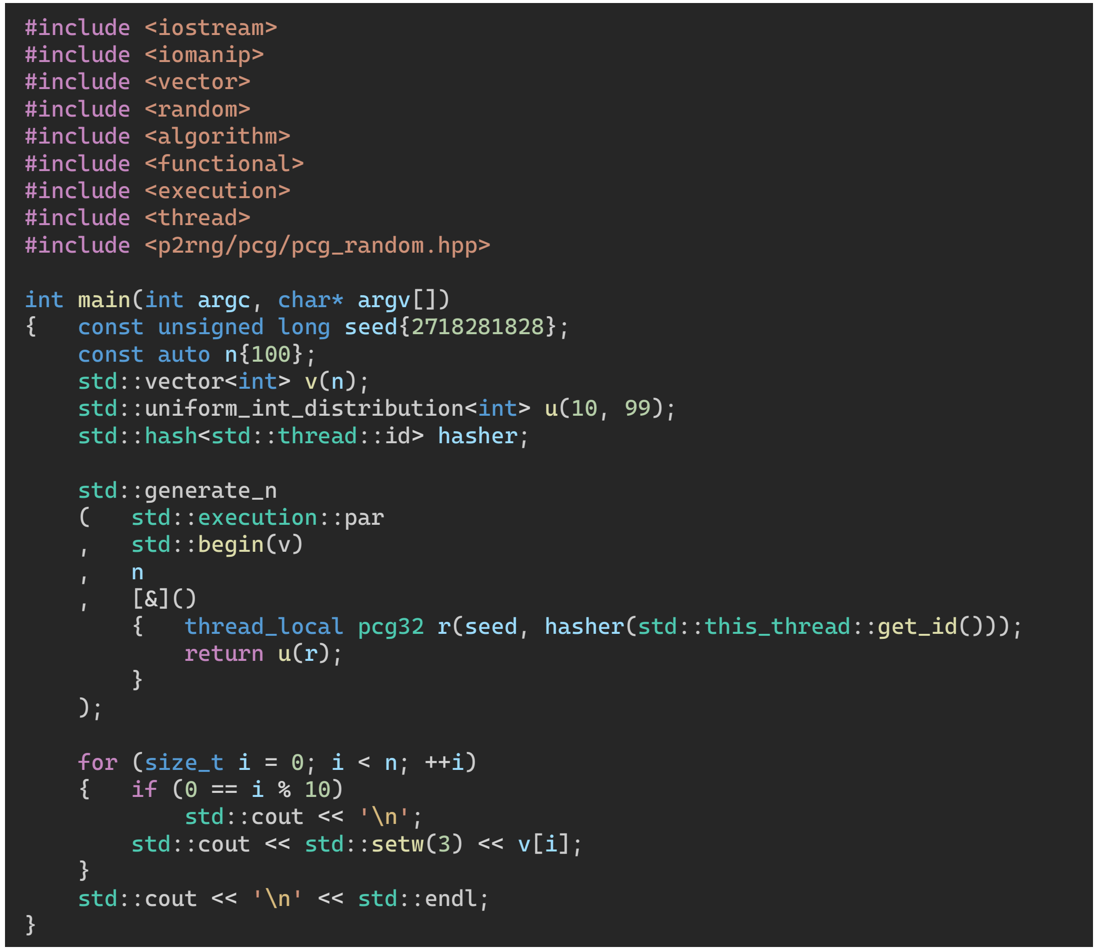
Pros

- Good scaling

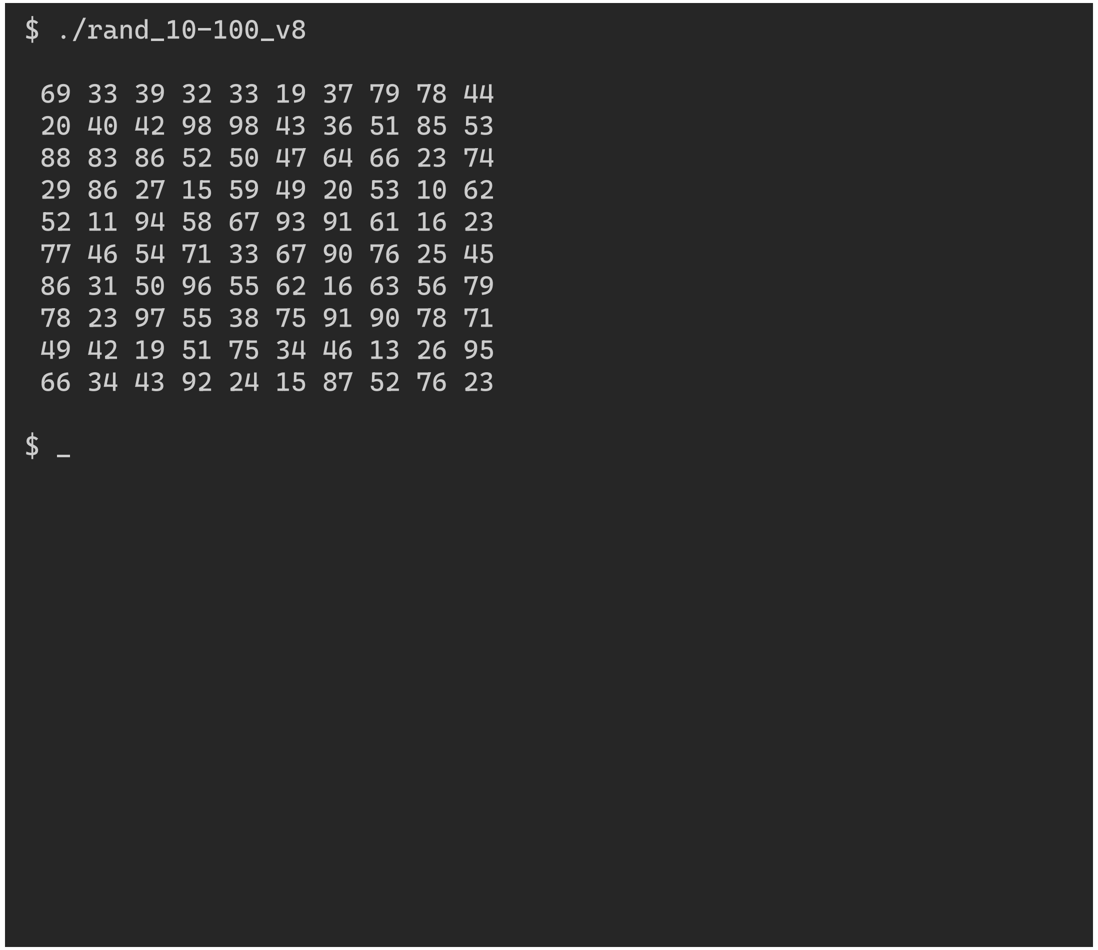
Cons

- Possible correlations in subsequences
- Overlapping of subsequences
- Cannot play fair

Parametrization (e.g. Changing Stream)

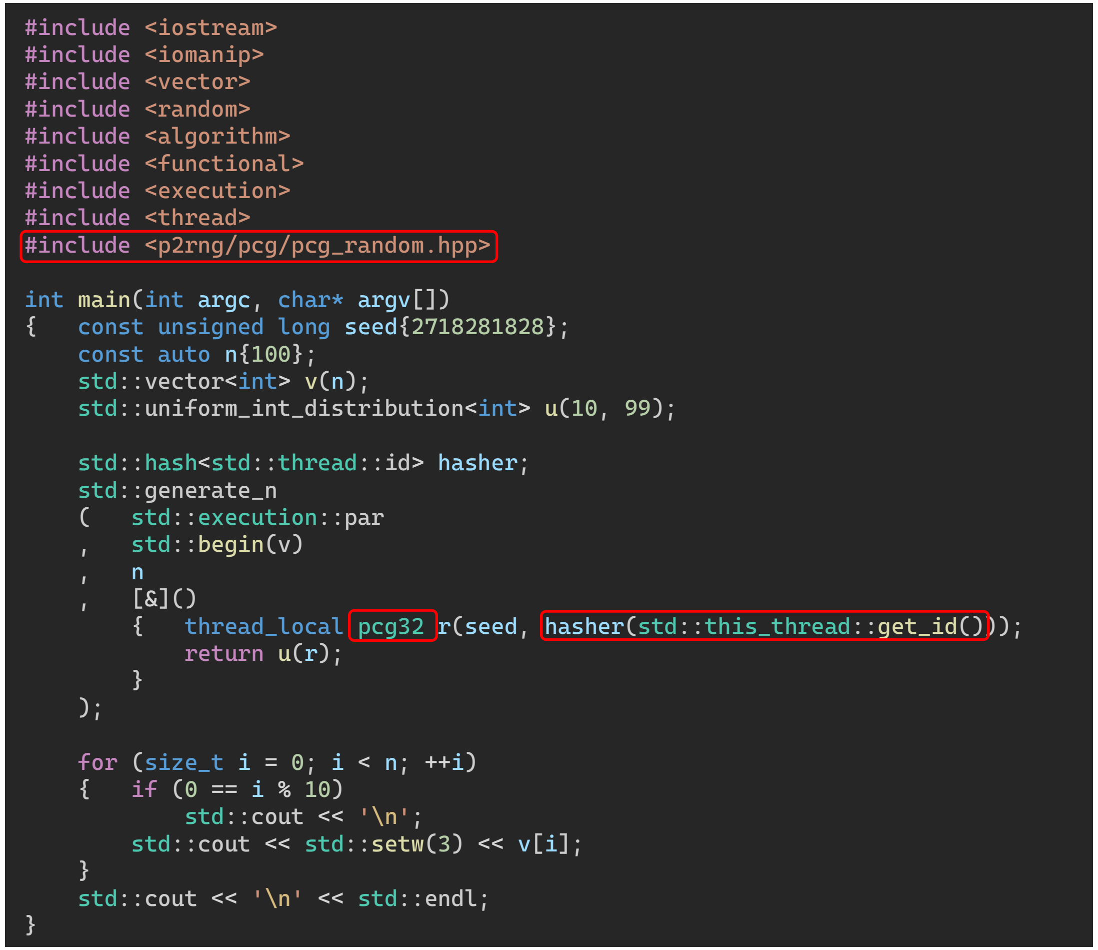


```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>
#include <execution>
#include <thread>
#include <p2rng/pcg/pcg_random.hpp>

int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::uniform_int_distribution<int> u(10, 99);
    std::hash<std::thread::id> hasher;

    std::generate_n
    (   std::execution::par
    ,   std::begin(v)
    ,   n
    ,   [&]()
        {   thread_local pcg32 r(seed, hasher(std::this_thread::get_id()));
            return u(r);
        }
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

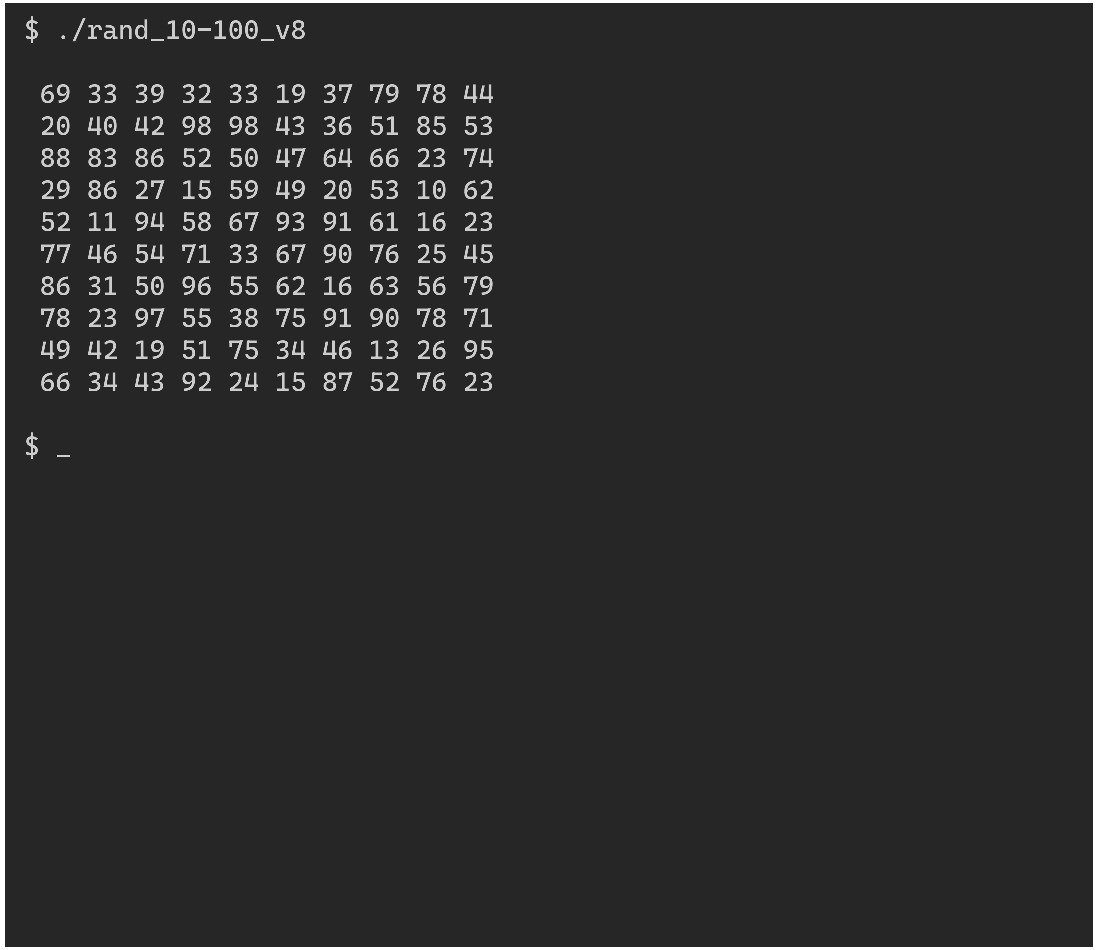


```
$ ./rand_10-100_v8

 69 33 39 32 33 19 37 79 78 44
 20 40 42 98 98 43 36 51 85 53
 88 83 86 52 50 47 64 66 23 74
 29 86 27 15 59 49 20 53 10 62
 52 11 94 58 67 93 91 61 16 23
 77 46 54 71 33 67 90 76 25 45
 86 31 50 96 55 62 16 63 56 79
 78 23 97 55 38 75 91 90 78 71
 49 42 19 51 75 34 46 13 26 95
 66 34 43 92 24 15 87 52 76 23

$ _
```

Parametrization (e.g. Changing Stream)

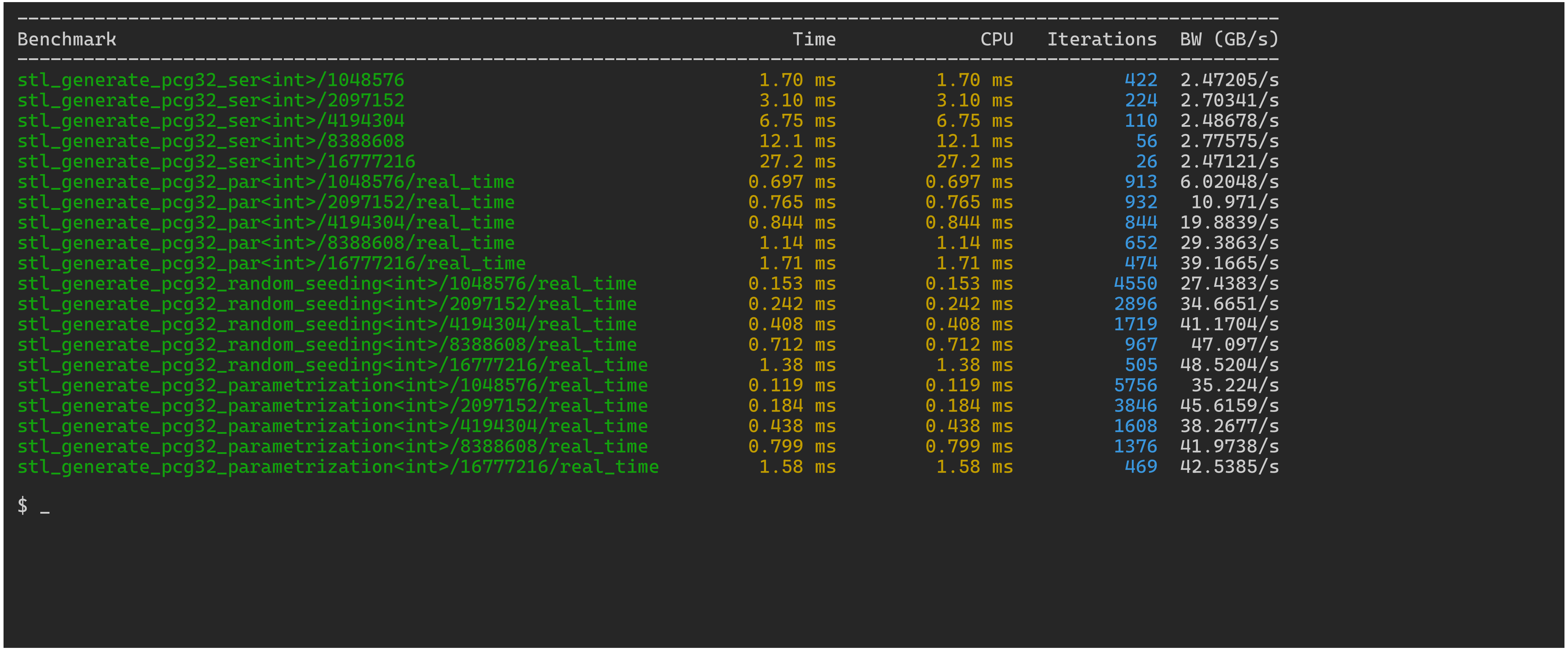


```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>
#include <execution>
#include <thread>
#include <p2rng/pcg/pcg_random.hpp>

int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::uniform_int_distribution<int> u(10, 99);

    std::hash<std::thread::id> hasher;
    std::generate_n
    (   std::execution::par
    ,   std::begin(v)
    ,   n
    ,   [&]()
        {   thread_local pcg32 r(seed, hasher(std::this_thread::get_id()));
            return u(r);
        }
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```



```
$ ./rand_10-100_v8

 69 33 39 32 33 19 37 79 78 44
 20 40 42 98 98 43 36 51 85 53
 88 83 86 52 50 47 64 66 23 74
 29 86 27 15 59 49 20 53 10 62
 52 11 94 58 67 93 91 61 16 23
 77 46 54 71 33 67 90 76 25 45
 86 31 50 96 55 62 16 63 56 79
 78 23 97 55 38 75 91 90 78 71
 49 42 19 51 75 34 46 13 26 95
 66 34 43 92 24 15 87 52 76 23

$ _
```

Parametrization – Benchmarks (AMD Epic 7543 CPU)



```
------------------------------------------------------------------------------------------------------------------
Benchmark                                                             Time             CPU   Iterations  BW (GB/s)
------------------------------------------------------------------------------------------------------------------
stl_generate_pcg32_ser<int>/1048576                                1.70 ms         1.70 ms          422  2.47205/s
stl_generate_pcg32_ser<int>/2097152                                3.10 ms         3.10 ms          224  2.70341/s
stl_generate_pcg32_ser<int>/4194304                                6.75 ms         6.75 ms          110  2.48678/s
stl_generate_pcg32_ser<int>/8388608                                12.1 ms         12.1 ms           56  2.77575/s
stl_generate_pcg32_ser<int>/16777216                               27.2 ms         27.2 ms           26  2.47121/s
stl_generate_pcg32_par<int>/1048576/real_time                     0.697 ms        0.697 ms          913  6.02048/s
stl_generate_pcg32_par<int>/2097152/real_time                     0.765 ms        0.765 ms          932   10.971/s
stl_generate_pcg32_par<int>/4194304/real_time                     0.844 ms        0.844 ms          844  19.8839/s
stl_generate_pcg32_par<int>/8388608/real_time                      1.14 ms         1.14 ms          652  29.3863/s
stl_generate_pcg32_par<int>/16777216/real_time                     1.71 ms         1.71 ms          474  39.1665/s
stl_generate_pcg32_random_seeding<int>/1048576/real_time          0.153 ms        0.153 ms         4550  27.4383/s
stl_generate_pcg32_random_seeding<int>/2097152/real_time          0.242 ms        0.242 ms         2896  34.6651/s
stl_generate_pcg32_random_seeding<int>/4194304/real_time          0.408 ms        0.408 ms         1719  41.1704/s
stl_generate_pcg32_random_seeding<int>/8388608/real_time          0.712 ms        0.712 ms          967   47.097/s
stl_generate_pcg32_random_seeding<int>/16777216/real_time          1.38 ms         1.38 ms          505  48.5204/s
stl_generate_pcg32_parametrization<int>/1048576/real_time         0.119 ms        0.119 ms         5756   35.224/s
stl_generate_pcg32_parametrization<int>/2097152/real_time         0.184 ms        0.184 ms         3846  45.6159/s
stl_generate_pcg32_parametrization<int>/4194304/real_time         0.438 ms        0.438 ms         1608  38.2677/s
stl_generate_pcg32_parametrization<int>/8388608/real_time         0.799 ms        0.799 ms         1376  41.9738/s
stl_generate_pcg32_parametrization<int>/16777216/real_time         1.58 ms         1.58 ms          469  42.5385/s

$ _
```

Parametrization – Verdict

Pros

- No overlapping

Cons

- Engine must support *multiple streams*
- Possible correlations between streams
- Cannot play fair

Leapfrog

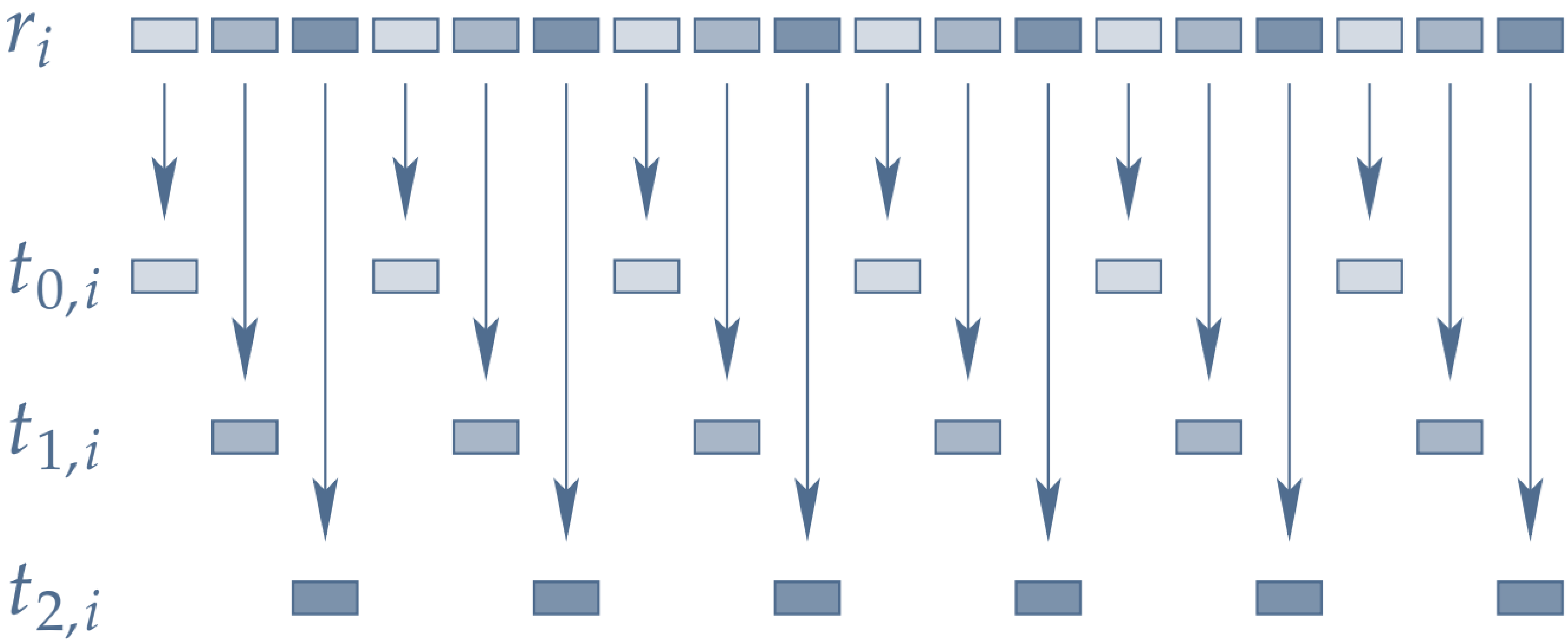


Image courtesy of https://github.com/rabauke/trng4

Leapfrog – Verdict

Pros

- No overlapping
- No correlations
- Can play fair

Cons

- The period is shortened by a factor of number-of-threads
- Prone to *false sharing*
- *Jump-ahead* feature must be supported by the engine
- Needs modified `generate()` algorithm

Block Splitting

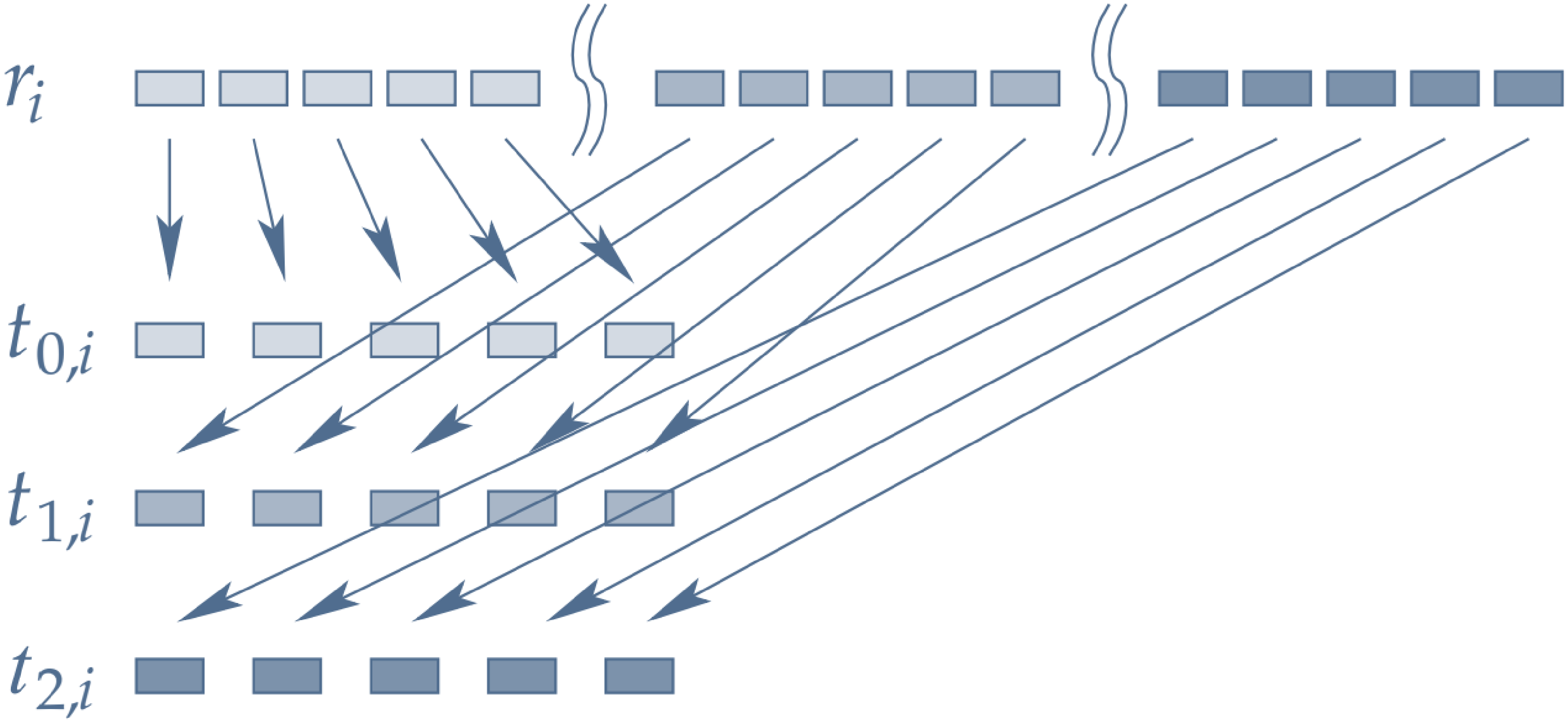


Image courtesy of https://github.com/rabauke/trng4

Block Splitting – Verdict

Pros

- No overlapping
- No correlations
- Can play fair

Cons

- The period is shortened by a factor of number-of-threads
- *Jump-ahead* feature must be supported by the engine
- Needs modified `generate()` algorithm

p2rng

Parallel Pseudo Random Number Generator

Ecosystems and Libraries with Parallel RNGs

Ecosystems

- CUDA – cuRAND
- ROCm – rocRAND
- oneAPI – oneMKL

Libraries

- SPRNG – http://sprng.org/
- TRNG4 – https://github.com/rabauke/trng4
- p2rng – https://github.com/arminms/p2rng

p2rng

https://github.com/arminms/p2rng

Features

- Multiplatform (Linux/macOS/Windows)
- Support four target APIs (OpenMP, CUDA, oneAPI, ROCm)
- Provide parallel versions of STL's std::generate() and std::generate_n() algorithms
- Play fair on all supported platforms
- Included engines: PCG Family (pcg-random.org)
- Included distributions: all 32 distributions provided by TRNG4 library
- Support CMake for building and auto configuration

Necessary and Sufficient Conditions to Play Fair

Must be supported by:

The algorithm	• Leapfrog • Block-splitting
The engine	• By providing an efficient jump ahead method (`discard(n)` function in C++11 standard)
The distribution	• By not discarding any values generated by the engine

`p2rng` fulfills all of them by:

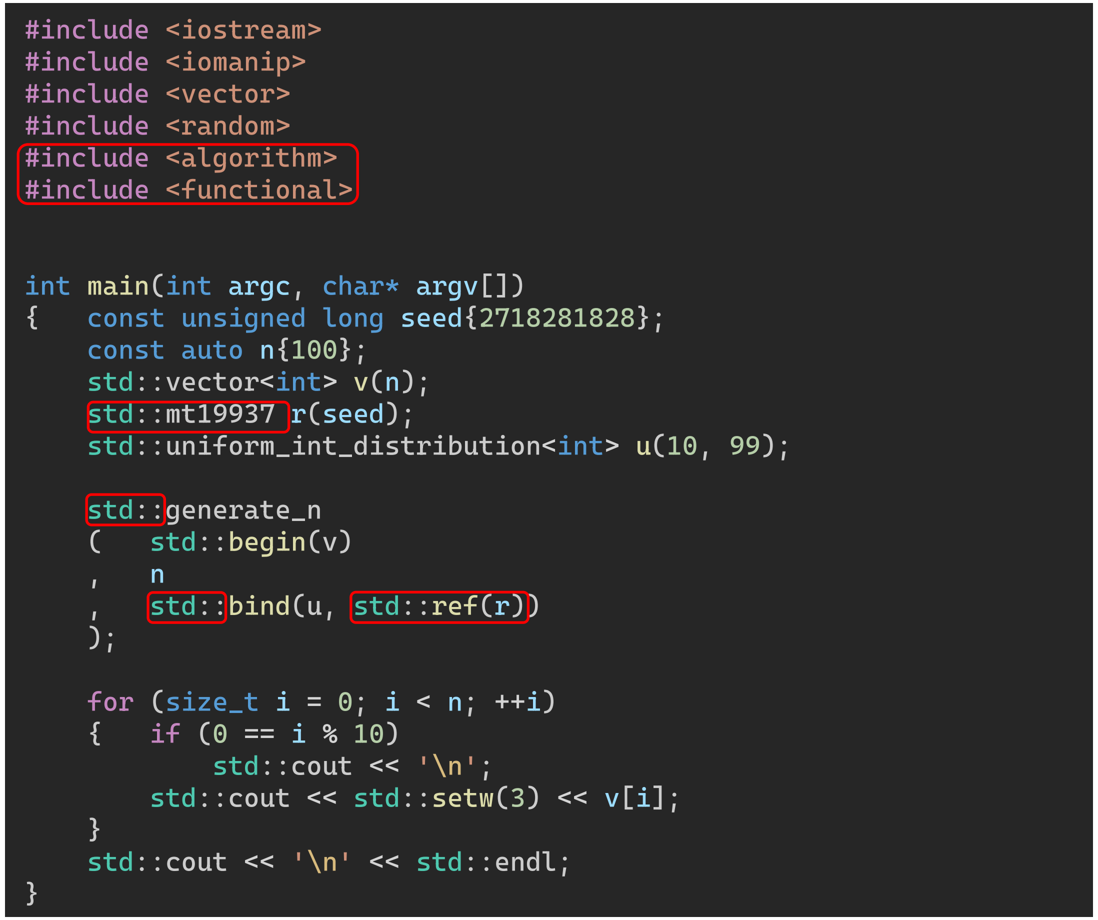
The algorithm	• Block-splitting
The engine	• *PCG Family* (variation of *LCG*) `discard(n)` takes *`O(Log n)`* time
The distribution	• Using the distributions provided by *TRNG4* library

Converting Serial STL Code to Parallel **`p2rng`**

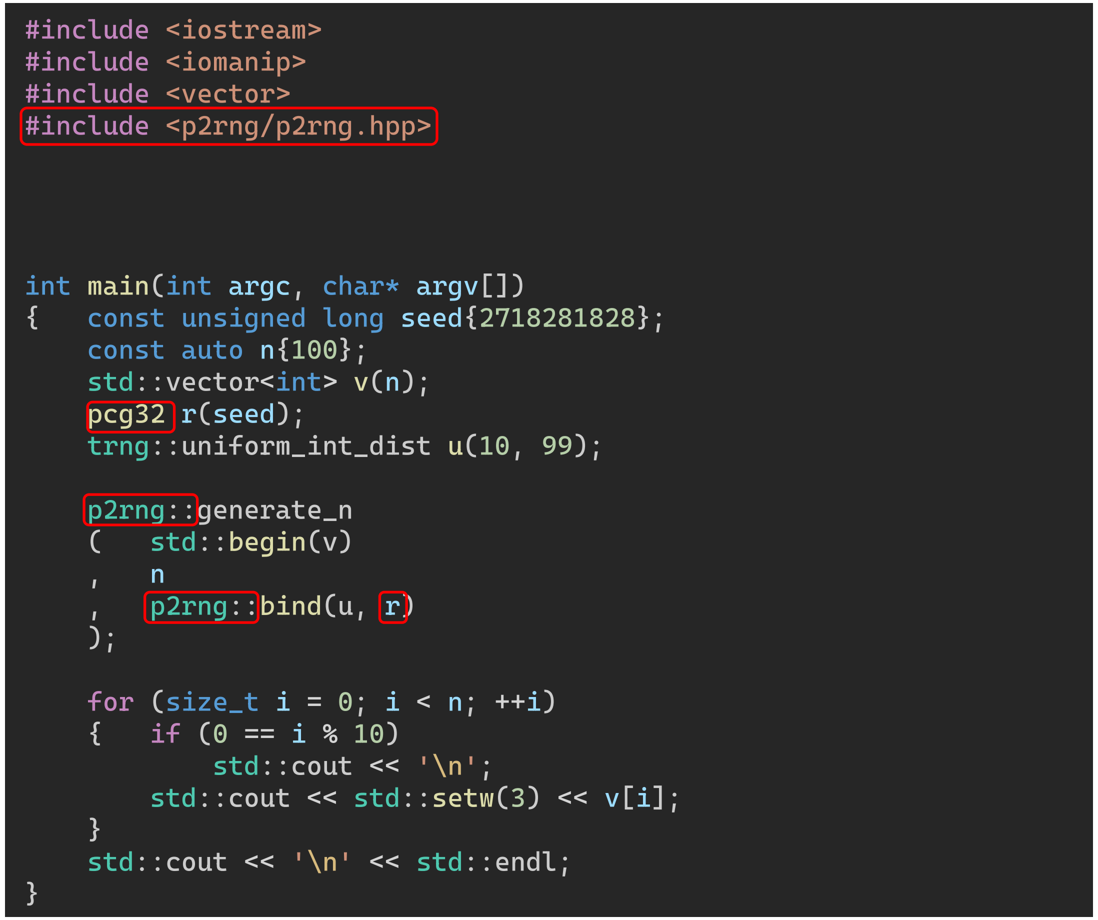
```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>


int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 99);

    std::generate_n
    (   std::begin(v)
    ,   n
    ,   std::bind(u, std::ref(r))
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <p2rng/p2rng.hpp>




int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    pcg32 r(seed);
    trng::uniform_int_dist u(10, 99);

    p2rng::generate_n
    (   std::begin(v)
    ,   n
    ,   p2rng::bind(u, r)
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

Converting Serial STL Code to Parallel **p2rng**

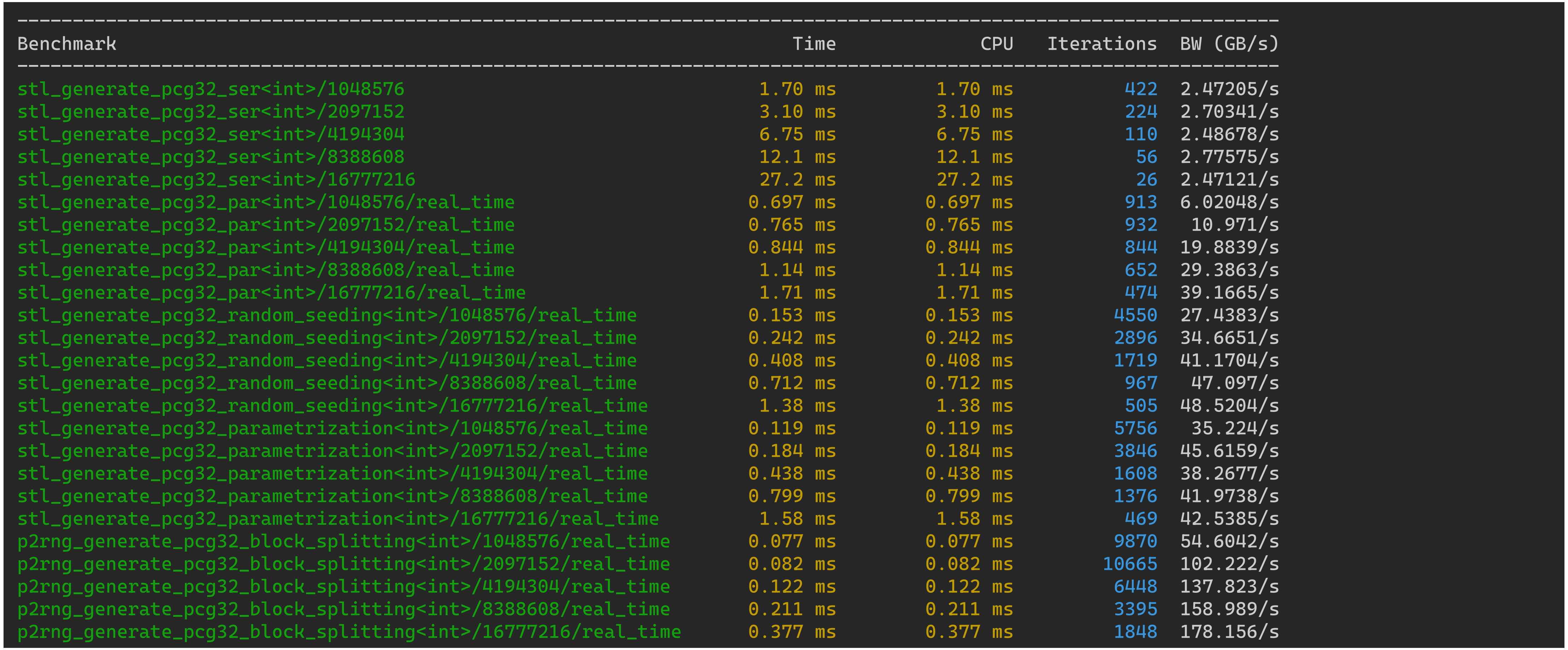


```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <random>
#include <algorithm>
#include <functional>


int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    std::mt19937 r(seed);
    std::uniform_int_distribution<int> u(10, 99);

    std::generate_n
    (   std::begin(v)
    ,   n
    ,   std::bind(u, std::ref(r))
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```



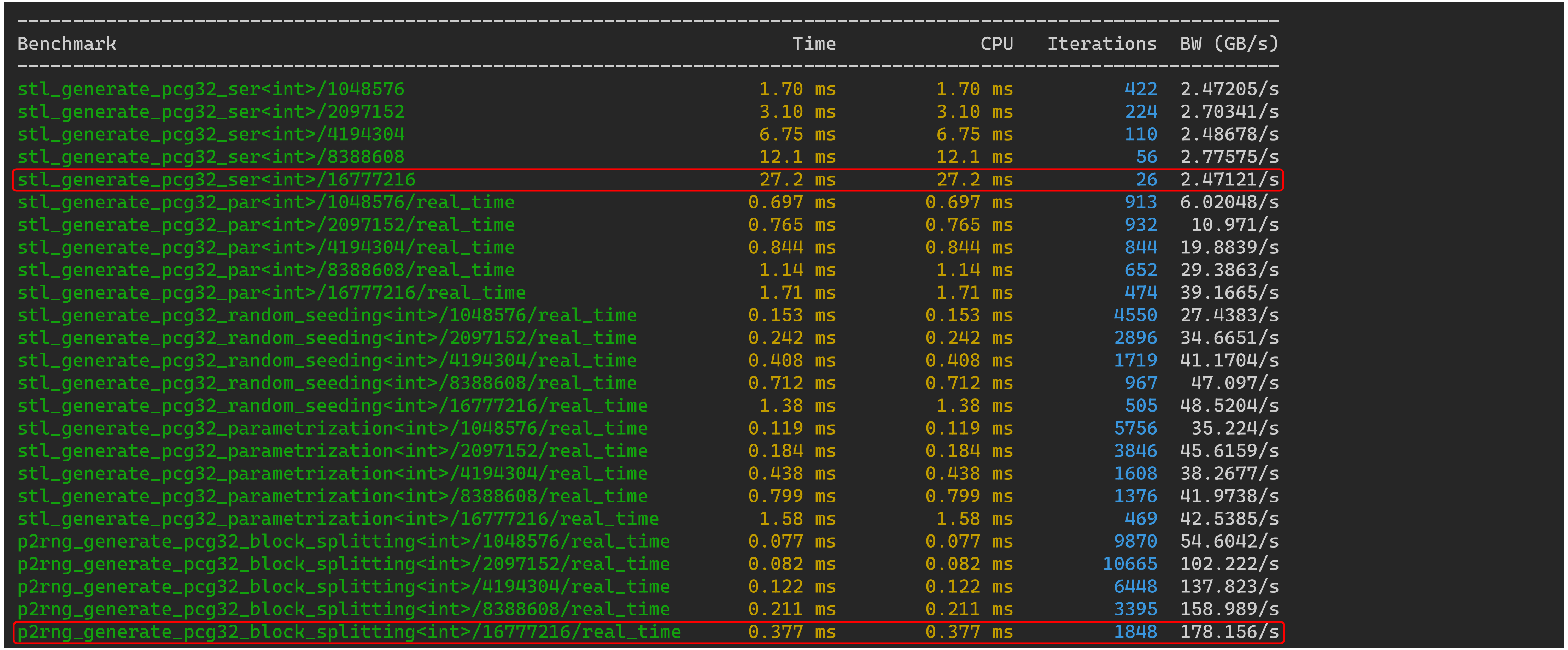
```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <p2rng/p2rng.hpp>



int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    pcg32 r(seed);
    trng::uniform_int_dist u(10, 99);

    p2rng::generate_n
    (   std::begin(v)
    ,   n
    ,   p2rng::bind(u, r)
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

p2rng Benchmarks – (AMD Epic 7543 CPU)



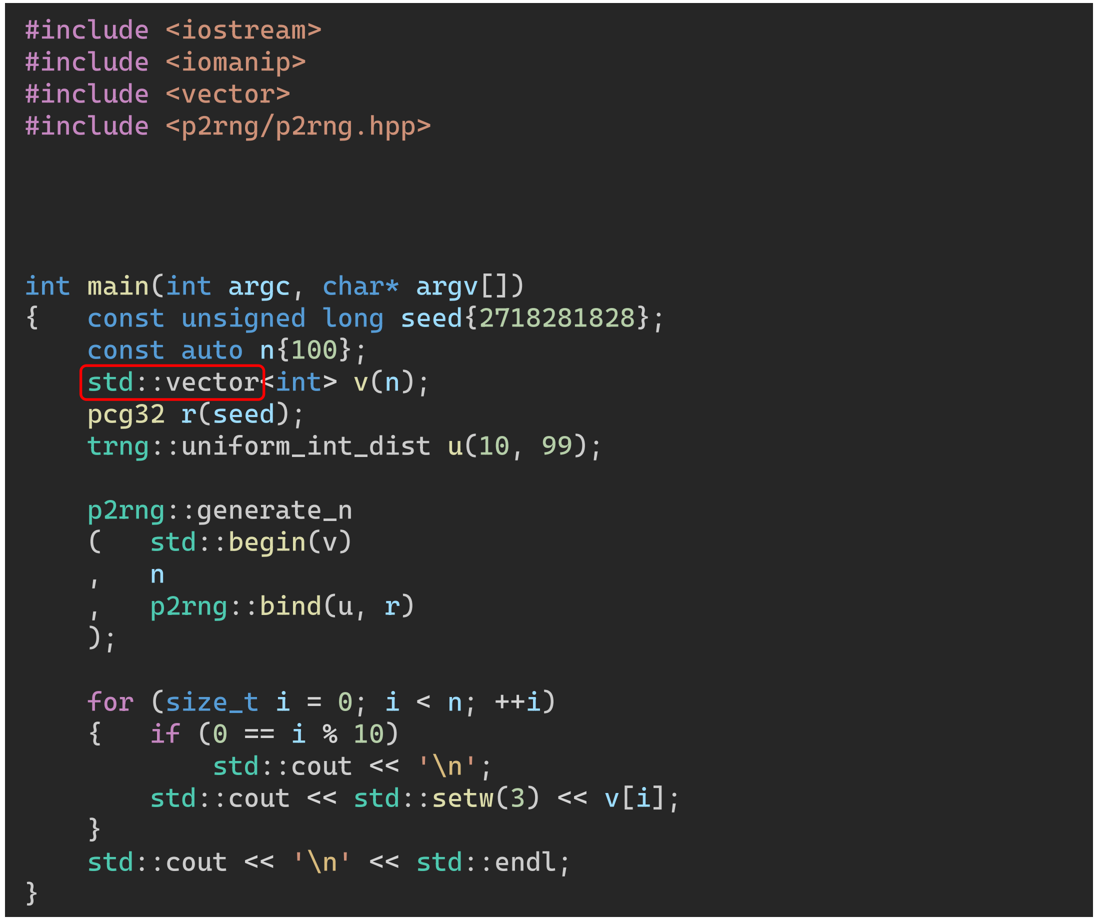
```
---------------------------------------------------------------------------------------------------------------------
Benchmark                                                                  Time             CPU   Iterations  BW (GB/s)
---------------------------------------------------------------------------------------------------------------------
stl_generate_pcg32_ser<int>/1048576                                     1.70 ms         1.70 ms          422  2.47205/s
stl_generate_pcg32_ser<int>/2097152                                     3.10 ms         3.10 ms          224  2.70341/s
stl_generate_pcg32_ser<int>/4194304                                     6.75 ms         6.75 ms          110  2.48678/s
stl_generate_pcg32_ser<int>/8388608                                     12.1 ms         12.1 ms           56  2.77575/s
stl_generate_pcg32_ser<int>/16777216                                    27.2 ms         27.2 ms           26  2.47121/s
stl_generate_pcg32_par<int>/1048576/real_time                          0.697 ms        0.697 ms          913  6.02048/s
stl_generate_pcg32_par<int>/2097152/real_time                          0.765 ms        0.765 ms          932   10.971/s
stl_generate_pcg32_par<int>/4194304/real_time                          0.844 ms        0.844 ms          844  19.8839/s
stl_generate_pcg32_par<int>/8388608/real_time                           1.14 ms         1.14 ms          652  29.3863/s
stl_generate_pcg32_par<int>/16777216/real_time                          1.71 ms         1.71 ms          474  39.1665/s
stl_generate_pcg32_random_seeding<int>/1048576/real_time               0.153 ms        0.153 ms         4550  27.4383/s
stl_generate_pcg32_random_seeding<int>/2097152/real_time               0.242 ms        0.242 ms         2896  34.6651/s
stl_generate_pcg32_random_seeding<int>/4194304/real_time               0.408 ms        0.408 ms         1719  41.1704/s
stl_generate_pcg32_random_seeding<int>/8388608/real_time               0.712 ms        0.712 ms          967   47.097/s
stl_generate_pcg32_random_seeding<int>/16777216/real_time               1.38 ms         1.38 ms          505  48.5204/s
stl_generate_pcg32_parametrization<int>/1048576/real_time              0.119 ms        0.119 ms         5756   35.224/s
stl_generate_pcg32_parametrization<int>/2097152/real_time              0.184 ms        0.184 ms         3846  45.6159/s
stl_generate_pcg32_parametrization<int>/4194304/real_time              0.438 ms        0.438 ms         1608  38.2677/s
stl_generate_pcg32_parametrization<int>/8388608/real_time              0.799 ms        0.799 ms         1376  41.9738/s
stl_generate_pcg32_parametrization<int>/16777216/real_time              1.58 ms         1.58 ms          469  42.5385/s
p2rng_generate_pcg32_block_splitting<int>/1048576/real_time            0.077 ms        0.077 ms         9870  54.6042/s
p2rng_generate_pcg32_block_splitting<int>/2097152/real_time            0.082 ms        0.082 ms        10665  102.222/s
p2rng_generate_pcg32_block_splitting<int>/4194304/real_time            0.122 ms        0.122 ms         6448  137.823/s
p2rng_generate_pcg32_block_splitting<int>/8388608/real_time            0.211 ms        0.211 ms         3395  158.989/s
p2rng_generate_pcg32_block_splitting<int>/16777216/real_time           0.377 ms        0.377 ms         1848  178.156/s
```

p2rng Benchmarks – (AMD Epic 7543 CPU)

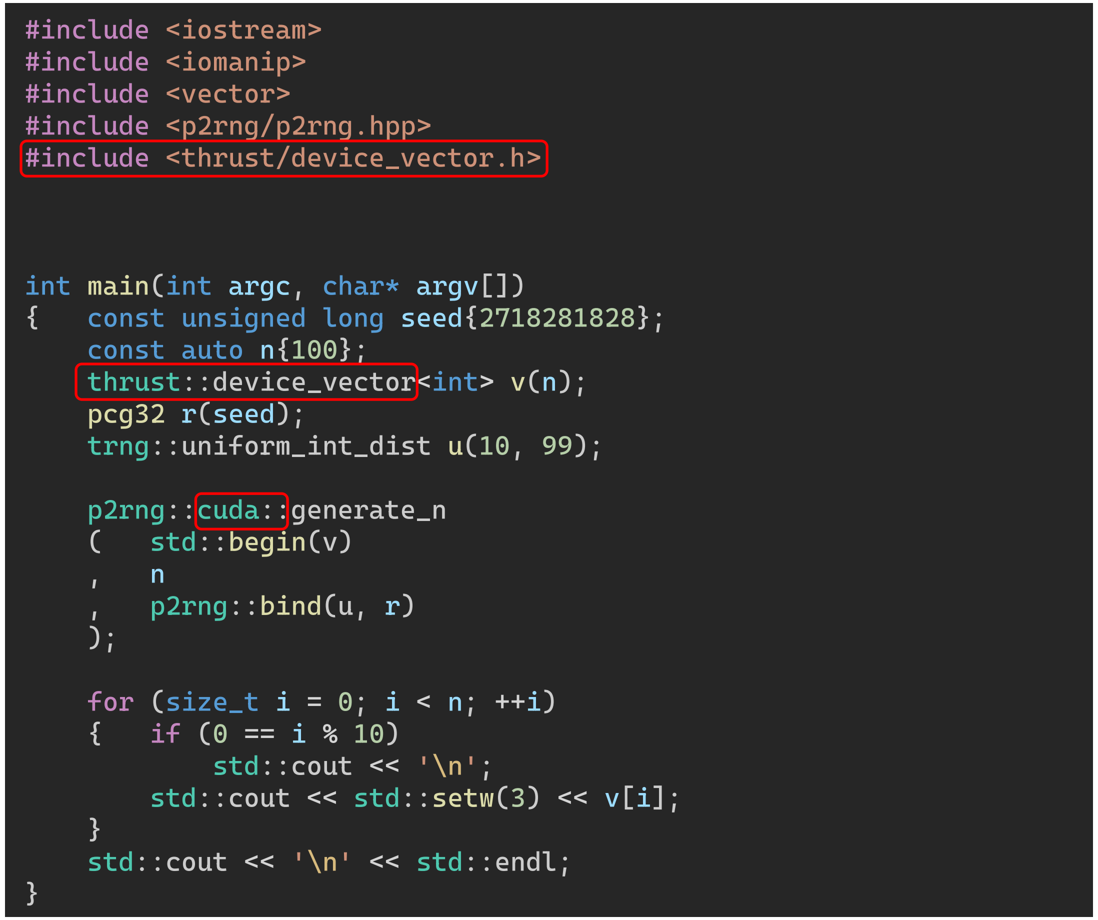


```
------------------------------------------------------------------------------------------------------------------
Benchmark                                                             Time             CPU   Iterations  BW (GB/s)
------------------------------------------------------------------------------------------------------------------
stl_generate_pcg32_ser<int>/1048576                                1.70 ms         1.70 ms          422  2.47205/s
stl_generate_pcg32_ser<int>/2097152                                3.10 ms         3.10 ms          224  2.70341/s
stl_generate_pcg32_ser<int>/4194304                                6.75 ms         6.75 ms          110  2.48678/s
stl_generate_pcg32_ser<int>/8388608                                12.1 ms         12.1 ms           56  2.77575/s
stl_generate_pcg32_ser<int>/16777216                               27.2 ms         27.2 ms           26  2.47121/s
stl_generate_pcg32_par<int>/1048576/real_time                     0.697 ms        0.697 ms          913  6.02048/s
stl_generate_pcg32_par<int>/2097152/real_time                     0.765 ms        0.765 ms          932   10.971/s
stl_generate_pcg32_par<int>/4194304/real_time                     0.844 ms        0.844 ms          844  19.8839/s
stl_generate_pcg32_par<int>/8388608/real_time                      1.14 ms         1.14 ms          652  29.3863/s
stl_generate_pcg32_par<int>/16777216/real_time                     1.71 ms         1.71 ms          474  39.1665/s
stl_generate_pcg32_random_seeding<int>/1048576/real_time          0.153 ms        0.153 ms         4550  27.4383/s
stl_generate_pcg32_random_seeding<int>/2097152/real_time          0.242 ms        0.242 ms         2896  34.6651/s
stl_generate_pcg32_random_seeding<int>/4194304/real_time          0.408 ms        0.408 ms         1719  41.1704/s
stl_generate_pcg32_random_seeding<int>/8388608/real_time          0.712 ms        0.712 ms          967   47.097/s
stl_generate_pcg32_random_seeding<int>/16777216/real_time          1.38 ms         1.38 ms          505  48.5204/s
stl_generate_pcg32_parametrization<int>/1048576/real_time         0.119 ms        0.119 ms         5756   35.224/s
stl_generate_pcg32_parametrization<int>/2097152/real_time         0.184 ms        0.184 ms         3846  45.6159/s
stl_generate_pcg32_parametrization<int>/4194304/real_time         0.438 ms        0.438 ms         1608  38.2677/s
stl_generate_pcg32_parametrization<int>/8388608/real_time         0.799 ms        0.799 ms         1376  41.9738/s
stl_generate_pcg32_parametrization<int>/16777216/real_time         1.58 ms         1.58 ms          469  42.5385/s
p2rng_generate_pcg32_block_splitting<int>/1048576/real_time       0.077 ms        0.077 ms         9870  54.6042/s
p2rng_generate_pcg32_block_splitting<int>/2097152/real_time       0.082 ms        0.082 ms        10665  102.222/s
p2rng_generate_pcg32_block_splitting<int>/4194304/real_time       0.122 ms        0.122 ms         6448  137.823/s
p2rng_generate_pcg32_block_splitting<int>/8388608/real_time       0.211 ms        0.211 ms         3395  158.989/s
p2rng_generate_pcg32_block_splitting<int>/16777216/real_time      0.377 ms        0.377 ms         1848  178.156/s
```

Converting OpenMP Code to CUDA/ROCm

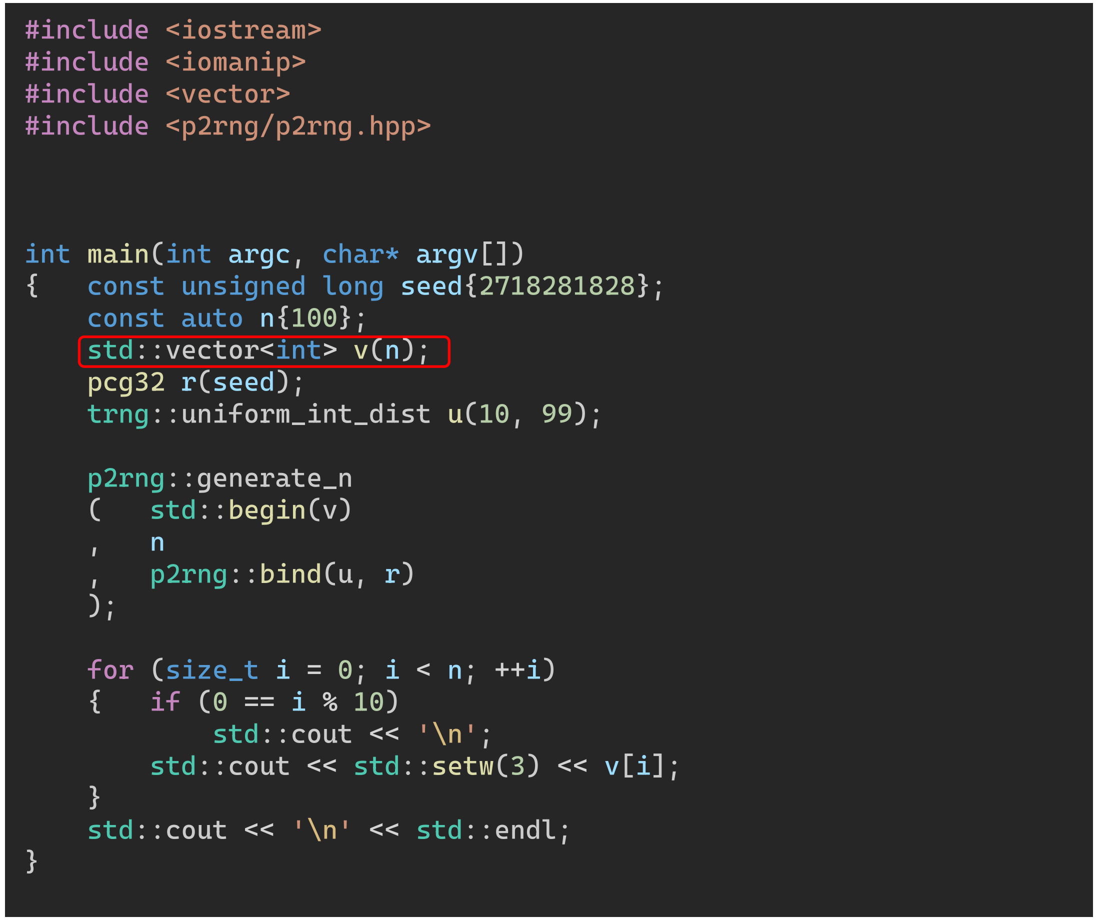


```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <p2rng/p2rng.hpp>



int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    pcg32 r(seed);
    trng::uniform_int_dist u(10, 99);

    p2rng::generate_n
    (   std::begin(v)
    ,   n
    ,   p2rng::bind(u, r)
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

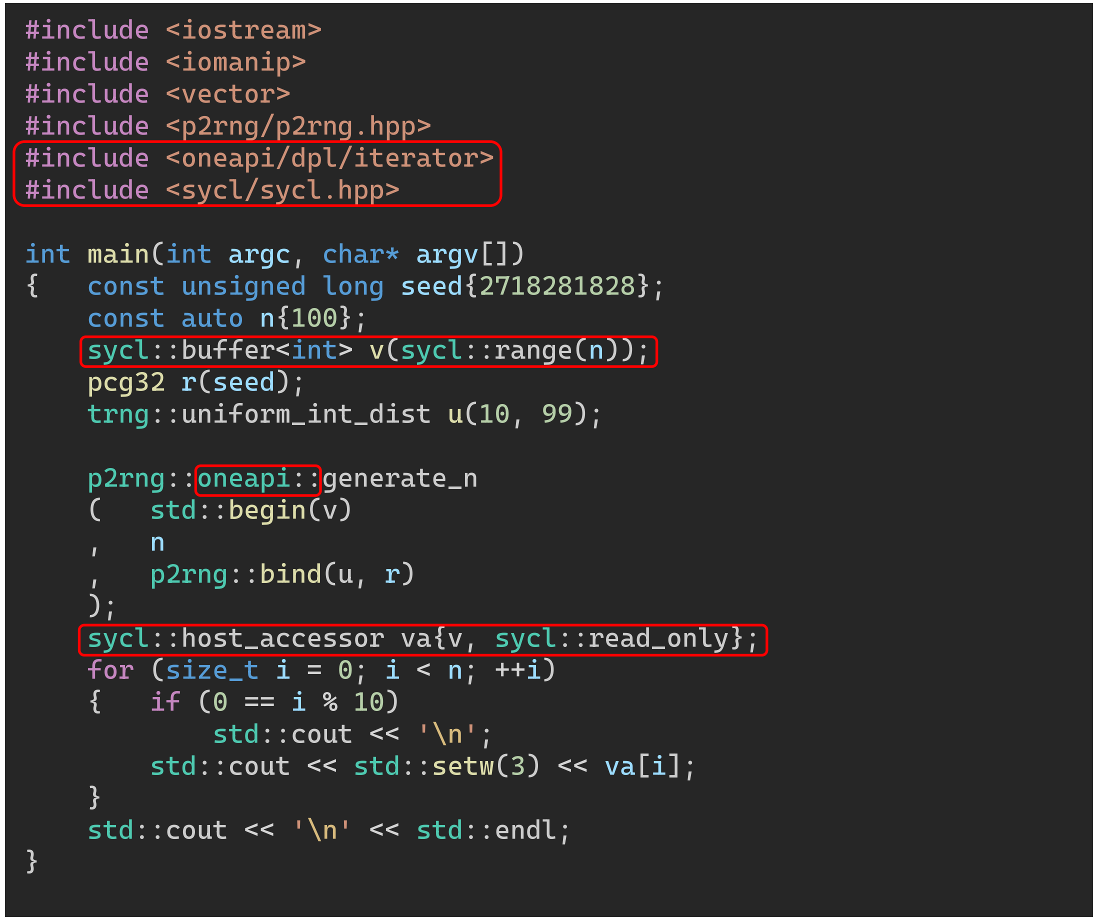


```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <p2rng/p2rng.hpp>
#include <thrust/device_vector.h>


int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    thrust::device_vector<int> v(n);
    pcg32 r(seed);
    trng::uniform_int_dist u(10, 99);

    p2rng::cuda::generate_n
    (   std::begin(v)
    ,   n
    ,   p2rng::bind(u, r)
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```

Converting OpenMP Code to oneAPI



```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <p2rng/p2rng.hpp>


int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    std::vector<int> v(n);
    pcg32 r(seed);
    trng::uniform_int_dist u(10, 99);

    p2rng::generate_n
    (   std::begin(v)
    ,   n
    ,   p2rng::bind(u, r)
    );

    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << v[i];
    }
    std::cout << '\n' << std::endl;
}
```



```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <p2rng/p2rng.hpp>
#include <oneapi/dpl/iterator>
#include <sycl/sycl.hpp>

int main(int argc, char* argv[])
{   const unsigned long seed{2718281828};
    const auto n{100};
    sycl::buffer<int> v(sycl::range(n));
    pcg32 r(seed);
    trng::uniform_int_dist u(10, 99);

    p2rng::oneapi::generate_n
    (   std::begin(v)
    ,   n
    ,   p2rng::bind(u, r)
    );
    sycl::host_accessor va{v, sycl::read_only};
    for (size_t i = 0; i < n; ++i)
    {   if (0 == i % 10)
            std::cout << '\n';
        std::cout << std::setw(3) << va[i];
    }
    std::cout << '\n' << std::endl;
}
```

Live Session

https://github.com/arminms/p2rng