# User

dll created using rust far bigger in size than g++

# Bing

The size of DLLs created using Rust can be larger than those created with G++ due to a few reasons:

1. **Static Linking**: By default, Rust statically links its standard library[**1**](https://stackoverflow.com/questions/29008127/why-are-rust-executables-so-huge)[**3**](https://stackoverflow.com/questions/28652031/why-does-rust-export-its-whole-standard-library-when-building-a-dll). This means that the entire standard library is included in the DLL, even if only a small part of it is used. This can significantly increase the size of the DLL[**2**](https://stackoverflow.com/questions/27325246/why-does-the-rust-compiler-generate-huge-executables).
2. **Optimization**: The Rust compiler optimizes for execution speed, compilation speed, and ease of debugging (by including symbols), rather than minimal binary size[**1**](https://stackoverflow.com/questions/29008127/why-are-rust-executables-so-huge).

However, there are ways to reduce the size of Rust binaries:

* Use Rust 1.32.0 or newer, which doesn't include jemalloc by default[**1**](https://stackoverflow.com/questions/29008127/why-are-rust-executables-so-huge).
* Add the following to Cargo.toml[**1**](https://stackoverflow.com/questions/29008127/why-are-rust-executables-so-huge):

[profile.release]

opt-level = 'z' # Optimize for size

lto = true # Enable link-time optimization

codegen-units = 1 # Reduce number of codegen units to increase optimizations

panic = 'abort' # Abort on panic

strip = true # Strip symbols from binary

* Compile with the -C prefer-dynamic option to dynamically link the standard library[**3**](https://stackoverflow.com/questions/28652031/why-does-rust-export-its-whole-standard-library-when-building-a-dll).
* Use the cdylib crate type when creating DLLs[**2**](https://stackoverflow.com/questions/27325246/why-does-the-rust-compiler-generate-huge-executables).

Remember, these methods might not be suitable for all use cases, and you should consider the trade-offs before applying them. For example, stripping symbols from the binary can make debugging more difficult. Also, dynamic linking can reduce the size of the binary, but it requires the linked libraries to be present on the system where the binary is run[**3**](https://stackoverflow.com/questions/28652031/why-does-rust-export-its-whole-standard-library-when-building-a-dll).

Source(s)

1. [Why are Rust executables so huge? - Stack Overflow](https://stackoverflow.com/questions/29008127/why-are-rust-executables-so-huge)

2. [Why does the Rust compiler generate huge executables?](https://stackoverflow.com/questions/27325246/why-does-the-rust-compiler-generate-huge-executables)

3. [Why does Rust export its whole standard library when building a DLL?](https://stackoverflow.com/questions/28652031/why-does-rust-export-its-whole-standard-library-when-building-a-dll)

4. [Creating A DLL With Rust - Sam Rambles](https://samrambles.com/guides/window-hacking-with-rust/creating-a-dll-with-rust/index.html)

5. <http://pastebin.com/LsG1u96C>

6. <http://pastebin.com/G0AYZrpF%29>

# User

But I am using cargo to build dll. using cargo buid --release -C prefer-dynamic produces errors:

error: unexpected argument '-C' found

Usage: cargo.exe build [OPTIONS]

For more information, try '--help'.

# Bing

I see, the -C prefer-dynamic option is a rustc flag, not a cargo flag. That's why you're seeing an error when you try to use it with cargo build.

To pass this flag to rustc through cargo, you need to use the RUSTFLAGS environment variable or specify it in a .cargo/config file[**1**](https://stackoverflow.com/questions/75022896/is-there-a-way-to-use-rust-cargo-to-link-some-of-the-dependency-crates-dynamical)[**3**](http://www.elebihan.com/posts/generating-dynamically-linked-programs-with-cargo.html)[**4**](https://stackoverflow.com/questions/47441765/why-is-the-rust-libstd-linked-dynamically-without-using-c-prefer-dynamic-flag-a). Here's how you can do it:

1. **Using RUSTFLAGS**: You can set the RUSTFLAGS environment variable in your terminal before running cargo build --release. Here's how you can do it in a Unix-like shell:

RUSTFLAGS="-C prefer-dynamic" cargo build --release

1. **Using .cargo/config**: You can create a .cargo/config file in your project root and specify the rustflags there[**1**](https://stackoverflow.com/questions/75022896/is-there-a-way-to-use-rust-cargo-to-link-some-of-the-dependency-crates-dynamical)[**3**](http://www.elebihan.com/posts/generating-dynamically-linked-programs-with-cargo.html)[**4**](https://stackoverflow.com/questions/47441765/why-is-the-rust-libstd-linked-dynamically-without-using-c-prefer-dynamic-flag-a). Here's an example:

[build]

rustflags = ["-C", "prefer-dynamic"]

After setting this, you can just run cargo build --release, and cargo will pass the -C prefer-dynamic option to rustc.

Remember, -C prefer-dynamic will cause rustc to prefer dynamic linking for all dependencies if possible[**1**](https://stackoverflow.com/questions/75022896/is-there-a-way-to-use-rust-cargo-to-link-some-of-the-dependency-crates-dynamical). If you want to link only some dependencies dynamically, you might need to adjust your project structure and use crate-type = ["dylib"] in the Cargo.toml of the crates you want to link dynamically[**1**](https://stackoverflow.com/questions/75022896/is-there-a-way-to-use-rust-cargo-to-link-some-of-the-dependency-crates-dynamical).

Source(s)

1. [dll - Is there a way to use Rust-Cargo to link SOME OF THE dependency ...](https://stackoverflow.com/questions/75022896/is-there-a-way-to-use-rust-cargo-to-link-some-of-the-dependency-crates-dynamical)

2. [rust - How can I use Cargo to build a library as both an rlib and a ...](https://stackoverflow.com/questions/40514793/how-can-i-use-cargo-to-build-a-library-as-both-an-rlib-and-a-dylib-but-with-diff)

3. [Generating Dynamically Linked Programs with Cargo](http://www.elebihan.com/posts/generating-dynamically-linked-programs-with-cargo.html)

4. [Why is the Rust libstd linked dynamically without using -C prefer ...](https://stackoverflow.com/questions/47441765/why-is-the-rust-libstd-linked-dynamically-without-using-c-prefer-dynamic-flag-a)

5. [Cargo prefers dynamic linking for a dylib, if in a workspace #7226 - GitHub](https://github.com/rust-lang/cargo/issues/7226)

6. <https://github.com/rust-lang/libc.git>

# User

I am on windows and it does not have .cargo folder in my user profile. Where to include above option?

# Bing

On Windows, the .cargo directory is typically located in your user profile directory, which is represented by the %USERPROFILE% environment variable[**1**](https://stackoverflow.com/questions/46649992/where-can-i-find-cargo-config)[**2**](https://doc.rust-lang.org/cargo/reference/config.html). If the .cargo directory doesn't exist, you can create it yourself[**1**](https://stackoverflow.com/questions/46649992/where-can-i-find-cargo-config).

Here's how you can create the .cargo directory and the config file:

1. Open a command prompt.
2. Run cd %USERPROFILE% to navigate to your user profile directory.
3. Run mkdir .cargo to create the .cargo directory.
4. Run cd .cargo to navigate into the .cargo directory.
5. Run echo [build] > config to create the config file with the [build] section.
6. Run echo rustflags = ["-C", "prefer-dynamic"] >> config to add the rustflags line to the config file.

After these steps, your config file in the .cargo directory should look like this:

[build]

rustflags = ["-C", "prefer-dynamic"]

Now, when you run cargo build --release in your project directory, cargo will pass the -C prefer-dynamic option to rustc[**1**](https://stackoverflow.com/questions/46649992/where-can-i-find-cargo-config)[**2**](https://doc.rust-lang.org/cargo/reference/config.html).

Remember, this will cause rustc to prefer dynamic linking for all dependencies if possible[**1**](https://stackoverflow.com/questions/46649992/where-can-i-find-cargo-config). If you want to link only some dependencies dynamically, you might need to adjust your project structure and use crate-type = ["dylib"] in the Cargo.toml of the crates you want to link dynamically[**1**](https://stackoverflow.com/questions/46649992/where-can-i-find-cargo-config).

Source(s)

1. [rust - Where can I find .cargo/config? - Stack Overflow](https://stackoverflow.com/questions/46649992/where-can-i-find-cargo-config)

2. [Configuration - The Cargo Book - Learn Rust](https://doc.rust-lang.org/cargo/reference/config.html)

3. [GitHub - microsoft/cargo-pkgconfig: pkg-config like interface for ...](https://github.com/microsoft/cargo-pkgconfig)

# User

Can I not include the -C flag at the time of compilation with Cargo Build --Release?

# Bing

The cargo build command does not directly accept the -C flag because it's a flag for the rustc compiler, not cargo. However, you can pass this flag to rustc through cargo using the RUSTFLAGS environment variable or by specifying it in a .cargo/config file.

Here's how you can do it:

**Using RUSTFLAGS**:
In a command prompt, you can set the RUSTFLAGS environment variable before running cargo build --release. Here's how you can do it:

set RUSTFLAGS=-C prefer-dynamic

cargo build --release

**Using .cargo/config**:
You can create a .cargo/config file in your project root and specify the rustflags there. Here's an example:

[build]

rustflags = ["-C", "prefer-dynamic"]

After setting this, you can just run cargo build --release, and cargo will pass the -C prefer-dynamic option to rustc.

Remember, -C prefer-dynamic will cause rustc to prefer dynamic linking for all dependencies if possible. If you want to link only some dependencies dynamically, you might need to adjust your project structure and use crate-type = ["dylib"] in the Cargo.toml of the crates you want to link dynamically.

# User

At present I have crate-type = ["cdylib"]. Do I change that?

# Bing

The crate-type you should use depends on your specific use case[**1**](https://users.rust-lang.org/t/what-is-the-difference-between-dylib-and-cdylib/28847)[**2**](https://doc.rust-lang.org/reference/linkage.html)[**3**](https://users.rust-lang.org/t/sharing-a-crate-with-crate-type-dylib/66768)[**4**](https://users.rust-lang.org/t/why-do-i-need-to-set-the-crate-type-to-cdylib-to-build-a-wasm-binary/93247)[**5**](https://stackoverflow.com/questions/44169635/rust-cargo-dylib-clarification).

* cdylib: This crate type is primarily designed for building shared libraries that can be linked into C/C++ programs[**1**](https://users.rust-lang.org/t/what-is-the-difference-between-dylib-and-cdylib/28847). It statically links the Rust dependencies, including the standard library[**1**](https://users.rust-lang.org/t/what-is-the-difference-between-dylib-and-cdylib/28847)[**5**](https://stackoverflow.com/questions/44169635/rust-cargo-dylib-clarification). This results in larger binary sizes, but the produced dynamic library is self-contained and can be used on systems that don't have the Rust standard library installed[**1**](https://users.rust-lang.org/t/what-is-the-difference-between-dylib-and-cdylib/28847)[**5**](https://stackoverflow.com/questions/44169635/rust-cargo-dylib-clarification).
* dylib: This crate type is used for dynamic linking with other Rust projects[**1**](https://users.rust-lang.org/t/what-is-the-difference-between-dylib-and-cdylib/28847)[**5**](https://stackoverflow.com/questions/44169635/rust-cargo-dylib-clarification). It does not statically link the Rust dependencies, so the produced dynamic library is smaller[**1**](https://users.rust-lang.org/t/what-is-the-difference-between-dylib-and-cdylib/28847). However, it requires the Rust standard library and any other dependencies to be present on the system where the binary is run[**1**](https://users.rust-lang.org/t/what-is-the-difference-between-dylib-and-cdylib/28847)[**5**](https://stackoverflow.com/questions/44169635/rust-cargo-dylib-clarification).

If you're building a shared library to be used from another language (like C or C++), cdylib is usually the better choice[**1**](https://users.rust-lang.org/t/what-is-the-difference-between-dylib-and-cdylib/28847)[**3**](https://users.rust-lang.org/t/sharing-a-crate-with-crate-type-dylib/66768)[**4**](https://users.rust-lang.org/t/why-do-i-need-to-set-the-crate-type-to-cdylib-to-build-a-wasm-binary/93247)[**5**](https://stackoverflow.com/questions/44169635/rust-cargo-dylib-clarification). If you're building a shared library to be used from other Rust projects, you might want to use dylib, but be aware that it can be a headache to work with outside of that[**3**](https://users.rust-lang.org/t/sharing-a-crate-with-crate-type-dylib/66768).

Remember, changing the crate-type can have significant effects on your project, so you should carefully consider the trade-offs before making a decision[**1**](https://users.rust-lang.org/t/what-is-the-difference-between-dylib-and-cdylib/28847)[**2**](https://doc.rust-lang.org/reference/linkage.html)[**3**](https://users.rust-lang.org/t/sharing-a-crate-with-crate-type-dylib/66768)[**4**](https://users.rust-lang.org/t/why-do-i-need-to-set-the-crate-type-to-cdylib-to-build-a-wasm-binary/93247)[**5**](https://stackoverflow.com/questions/44169635/rust-cargo-dylib-clarification). If you're unsure, you might want to ask for advice on a Rust community forum or Stack Overflow[**1**](https://users.rust-lang.org/t/what-is-the-difference-between-dylib-and-cdylib/28847)[**2**](https://doc.rust-lang.org/reference/linkage.html)[**3**](https://users.rust-lang.org/t/sharing-a-crate-with-crate-type-dylib/66768)[**4**](https://users.rust-lang.org/t/why-do-i-need-to-set-the-crate-type-to-cdylib-to-build-a-wasm-binary/93247)[**5**](https://stackoverflow.com/questions/44169635/rust-cargo-dylib-clarification).

Source(s)

1. [What is the Difference Between `dylib` and `cdylib`](https://users.rust-lang.org/t/what-is-the-difference-between-dylib-and-cdylib/28847)

2. [Linkage - The Rust Reference](https://doc.rust-lang.org/reference/linkage.html)

3. [Sharing a crate with crate-type = dylib - help - The Rust Programming ...](https://users.rust-lang.org/t/sharing-a-crate-with-crate-type-dylib/66768)

4. [Why do I need to set the crate-type to cdylib to build a wasm binary?](https://users.rust-lang.org/t/why-do-i-need-to-set-the-crate-type-to-cdylib-to-build-a-wasm-binary/93247)

5. [macos - Rust cargo dylib clarification - Stack Overflow](https://stackoverflow.com/questions/44169635/rust-cargo-dylib-clarification)

# User

Can you tell me whether following rust code (compiled into dll) indeed use 4 cores of cpu and does parallel processing in getting random numbers. Please note that input arguments are number of threads (corresponding to number of cpu cores) and number of random numbers to be generated.

#![feature(core\_intrinsics)] // Enable core intrinsics for \_rdrand64\_step

use std::alloc::{alloc, dealloc, Layout};
use core::arch::x86\_64::\_rdrand64\_step;

extern crate rand; // Import the rand crate
use rand::Rng;

#[no\_mangle]
pub extern "C" fn rdrand64\_step(rand: &mut u64) -> i32 {
unsafe {
if \_rdrand64\_step(rand) == 1 {
// Check if the number has exactly 15 digits
if \*rand >= 100\_000\_000\_000\_000 && \*rand <= 999\_999\_999\_999\_999 {
return 1; // Success
}
}
// Fallback to software RNG if hardware RNG fails or does not produce a 15-digit number
\*rand = rand::thread\_rng().gen\_range(100\_000\_000\_000\_000..=999\_999\_999\_999\_999);
return 0; // Indicate fallback
}
}

#[no\_mangle]
pub extern "C" fn generate\_random\_numbers(num\_threads: i32, num\_numbers: i32) {
unsafe {
let mut rand: u64 = 0;
for thread\_num in 0..num\_threads {
for i in 0..num\_numbers / num\_threads {
rdrand64\_step(&mut rand);
\*NUMBERS.offset((thread\_num \* num\_numbers / num\_threads + i) as isize) = rand;
}
}
}
}

static mut NUMBERS: \*mut u64 = std::ptr::null\_mut();

#[no\_mangle]
pub extern "C" fn get\_numbers() -> \*mut u64 {
unsafe { NUMBERS }
}

#[no\_mangle]
pub extern "C" fn allocate\_memory(num\_numbers: i32) {
unsafe {
let layout = Layout::from\_size\_align(num\_numbers as usize \* std::mem::size\_of::<u64>(), std::mem::align\_of::<u64>()).unwrap();
NUMBERS = alloc(layout) as \*mut u64;
}
}

#[no\_mangle]
pub extern "C" fn free\_memory() {
unsafe {
let layout = Layout::from\_size\_align(std::mem::size\_of\_val(&\*NUMBERS), std::mem::align\_of\_val(&\*NUMBERS)).unwrap();
dealloc(NUMBERS as \*mut u8, layout);
}
}

# Bing

The Rust code you've provided does not inherently use parallel processing or multiple cores of the CPU. The generate\_random\_numbers function, which generates random numbers, runs in a single thread[**1**](https://rust-lang-nursery.github.io/rust-cookbook/concurrency/parallel.html)[**2**](https://engineering.deptagency.com/parallel-processing-in-rust)[**3**](https://marketsplash.com/tutorials/rust/rust-parallelism/).

Although the function takes num\_threads as an argument and uses it in a loop, it doesn't actually create or use multiple threads. The loop with thread\_num simply iterates num\_threads times sequentially in the same thread[**1**](https://rust-lang-nursery.github.io/rust-cookbook/concurrency/parallel.html)[**2**](https://engineering.deptagency.com/parallel-processing-in-rust)[**3**](https://marketsplash.com/tutorials/rust/rust-parallelism/).

To utilize multiple cores and perform parallel processing in Rust, you would typically use a crate like rayon[**1**](https://rust-lang-nursery.github.io/rust-cookbook/concurrency/parallel.html)[**3**](https://marketsplash.com/tutorials/rust/rust-parallelism/). This crate provides simple and efficient parallelism by turning sequential iterators into parallel ones[**1**](https://rust-lang-nursery.github.io/rust-cookbook/concurrency/parallel.html)[**3**](https://marketsplash.com/tutorials/rust/rust-parallelism/).

Here's a simplified example of how you might use rayon to generate random numbers in parallel:

use rand::Rng;

use rayon::prelude::\*;

fn generate\_random\_numbers(num\_threads: usize, num\_numbers: usize) -> Vec<u64> {

(0..num\_threads).into\_par\_iter().flat\_map(|\_| {

let mut rng = rand::thread\_rng();

(0..num\_numbers).map(|\_| rng.gen\_range(100\_000\_000\_000\_000..=999\_999\_999\_999\_999)).collect::<Vec<\_>>()

}).collect()

}

In this example, into\_par\_iter creates a parallel iterator, and flat\_map applies a function to each element of the iterator and flattens the results into a single, flat sequence[**1**](https://rust-lang-nursery.github.io/rust-cookbook/concurrency/parallel.html)[**3**](https://marketsplash.com/tutorials/rust/rust-parallelism/). This allows the random number generation to be performed in parallel across multiple threads[**1**](https://rust-lang-nursery.github.io/rust-cookbook/concurrency/parallel.html)[**3**](https://marketsplash.com/tutorials/rust/rust-parallelism/).

Remember, parallel processing can speed up computation, but it also adds complexity and overhead. It's most beneficial when the computations are heavy and can be performed independently[**1**](https://rust-lang-nursery.github.io/rust-cookbook/concurrency/parallel.html)[**2**](https://engineering.deptagency.com/parallel-processing-in-rust)[**3**](https://marketsplash.com/tutorials/rust/rust-parallelism/). Always consider the trade-offs before deciding to use parallelism[**1**](https://rust-lang-nursery.github.io/rust-cookbook/concurrency/parallel.html)[**2**](https://engineering.deptagency.com/parallel-processing-in-rust)[**3**](https://marketsplash.com/tutorials/rust/rust-parallelism/).

Source(s)

1. [Data Parallelism - Rust Cookbook - GitHub Pages](https://rust-lang-nursery.github.io/rust-cookbook/concurrency/parallel.html)

2. [Getting Started with Parallel Processing in Rust](https://engineering.deptagency.com/parallel-processing-in-rust)

3. [Rust Parallel Programming: Concurrency For Faster Execution - MarketSplash](https://marketsplash.com/tutorials/rust/rust-parallelism/)

4. [Faster compilation with the parallel front-end in nightly | Rust Blog](https://blog.rust-lang.org/2023/11/09/parallel-rustc.html?ref=upstract.com)

5. [Back-end parallelism in the Rust compiler | Nicholas Nethercote](https://nnethercote.github.io/2023/07/11/back-end-parallelism-in-the-rust-compiler.html)

# User

Can you modify and give me complete code to process in parallel?

# Bing

Sure, I can help you modify your code to use parallel processing with the rayon crate. Here's an example of how you might do it:

#![feature(core\_intrinsics)] // Enable core intrinsics for \_rdrand64\_step

use std::alloc::{alloc, dealloc, Layout};

use core::arch::x86\_64::\_rdrand64\_step;

use rayon::prelude::\*; // Import the rayon crate for parallel processing

extern crate rand; // Import the rand crate

use rand::Rng;

#[no\_mangle]

pub extern "C" fn rdrand64\_step(rand: &mut u64) -> i32 {

unsafe {

if \_rdrand64\_step(rand) == 1 {

// Check if the number has exactly 15 digits

if \*rand >= 100\_000\_000\_000\_000 && \*rand <= 999\_999\_999\_999\_999 {

return 1; // Success

}

}

// Fallback to software RNG if hardware RNG fails or does not produce a 15-digit number

\*rand = rand::thread\_rng().gen\_range(100\_000\_000\_000\_000..=999\_999\_999\_999\_999);

return 0; // Indicate fallback

}

}

#[no\_mangle]

pub extern "C" fn generate\_random\_numbers(num\_threads: i32, num\_numbers: i32) {

let numbers\_per\_thread = num\_numbers / num\_threads;

let mut rng = rand::thread\_rng();

(0..num\_threads).into\_par\_iter().for\_each(|\_| {

for \_ in 0..numbers\_per\_thread {

let mut rand: u64 = 0;

rdrand64\_step(&mut rand);

unsafe {

\*NUMBERS.offset((thread\_num \* numbers\_per\_thread + i) as isize) = rand;

}

}

});

}

static mut NUMBERS: \*mut u64 = std::ptr::null\_mut();

#[no\_mangle]

pub extern "C" fn get\_numbers() -> \*mut u64 {

unsafe { NUMBERS }

}

#[no\_mangle]

pub extern "C" fn allocate\_memory(num\_numbers: i32) {

unsafe {

let layout = Layout::from\_size\_align(num\_numbers as usize \* std::mem::size\_of::<u64>(), std::mem::align\_of::<u64>()).unwrap();

NUMBERS = alloc(layout) as \*mut u64;

}

}

#[no\_mangle]

pub extern "C" fn free\_memory() {

unsafe {

let layout = Layout::from\_size\_align(std::mem::size\_of\_val(&\*NUMBERS), std::mem::align\_of\_val(&\*NUMBERS)).unwrap();

dealloc(NUMBERS as \*mut u8, layout);

}

}

In this modified version of your code, I've used the rayon crate's into\_par\_iter method to create a parallel iterator from the range of thread numbers. The for\_each method applies a closure to each item in the parallel iterator, in parallel.

Please note that this is a simplified example and may not work perfectly for your specific use case. You might need to adjust it to fit your needs. Also, remember to add rayon to your Cargo.toml dependencies.

# User

Please tell me how to add rayon crate and include it in Cargo.toml.

# Bing

To add the rayon crate to your Rust project, you need to include it in your Cargo.toml file[**1**](https://stackoverflow.com/questions/36808868/cant-find-crate-for-rayon)[**2**](https://docs.rs/crate/rayon/latest). Here's how you can do it:

1. Open your Cargo.toml file in a text editor.
2. Add the following lines to the [dependencies] section:

[dependencies]

rayon = "1.8"

The "1.8" specifies the version of rayon you want to use[**2**](https://docs.rs/crate/rayon/latest). You can replace this with the latest version if you prefer[**2**](https://docs.rs/crate/rayon/latest).

1. Save and close the Cargo.toml file.

Now, cargo will automatically download and compile the rayon crate when you build your project[**1**](https://stackoverflow.com/questions/36808868/cant-find-crate-for-rayon)[**2**](https://docs.rs/crate/rayon/latest).

If you don't have a Cargo.toml file, you can create one by initializing a new Cargo project with the command cargo init[**1**](https://stackoverflow.com/questions/36808868/cant-find-crate-for-rayon). This will create a new Cargo project in the current directory, complete with a Cargo.toml file[**1**](https://stackoverflow.com/questions/36808868/cant-find-crate-for-rayon).

Remember, you need to import the rayon crate in your Rust files to use it. You can do this by adding the following line at the top of your Rust files:

use rayon::prelude::\*;

This line imports the rayon prelude, which includes the traits and types needed to use rayon's parallel iterators[**2**](https://docs.rs/crate/rayon/latest).

Source(s)

1. [rust - Can't find crate for `rayon` - Stack Overflow](https://stackoverflow.com/questions/36808868/cant-find-crate-for-rayon)

2. [rayon 1.8.1 - Docs.rs](https://docs.rs/crate/rayon/latest)

3. [rayon 1.7.0 on Cargo - Libraries.io](https://libraries.io/cargo/rayon)

# User

How to first add rayon in my existing rust installation setup?

# Bing

To add the rayon crate to your existing Rust setup, you need to include it in your Cargo.toml file[**1**](https://docs.rs/rayon/)[**3**](https://stackoverflow.com/questions/36808868/cant-find-crate-for-rayon). Here's how you can do it:

1. Open your Cargo.toml file in a text editor.
2. Add the following lines to the [dependencies] section:

[dependencies]

rayon = "1.8"

The "1.8" specifies the version of rayon you want to use[**1**](https://docs.rs/rayon/)[**3**](https://stackoverflow.com/questions/36808868/cant-find-crate-for-rayon). You can replace this with the latest version if you prefer[**1**](https://docs.rs/rayon/)[**3**](https://stackoverflow.com/questions/36808868/cant-find-crate-for-rayon).

1. Save and close the Cargo.toml file.

Now, cargo will automatically download and compile the rayon crate when you build your project[**1**](https://docs.rs/rayon/)[**3**](https://stackoverflow.com/questions/36808868/cant-find-crate-for-rayon).

If you don't have a Cargo.toml file, you can create one by initializing a new Cargo project with the command cargo init[**1**](https://docs.rs/rayon/). This will create a new Cargo project in the current directory, complete with a Cargo.toml file[**1**](https://docs.rs/rayon/).

Remember, you need to import the rayon crate in your Rust files to use it. You can do this by adding the following line at the top of your Rust files:

use rayon::prelude::\*;

This line imports the rayon prelude, which includes the traits and types needed to use rayon's parallel iterators[**1**](https://docs.rs/rayon/)[**3**](https://stackoverflow.com/questions/36808868/cant-find-crate-for-rayon).

Source(s)

1. [rayon - Rust - Docs.rs](https://docs.rs/rayon/)

2. [rayon 1.8.1 - Docs.rs](https://docs.rs/crate/rayon/latest)

3. [rust - Can't find crate for `rayon` - Stack Overflow](https://stackoverflow.com/questions/36808868/cant-find-crate-for-rayon)

4. [rayon - Rust - GitHub Pages](https://friktor.github.io/jss-rs/rayon/index.html)

# User

Compilation errors:

error[E0425]: cannot find value thread\_num in this scope
--> src/rust\_rand\_dll\_copilot\_parallel.rs:34:34
|
34 | \*NUMBERS.offset((thread\_num \* numbers\_per\_thread + i) as isize) = rand;
| ^^^^^^^^^^ not found in this scope

error[E0425]: cannot find value i in this scope
--> src/rust\_rand\_dll\_copilot\_parallel.rs:34:68
|
34 | \*NUMBERS.offset((thread\_num \* numbers\_per\_thread + i) as isize) = rand;
| ^ not found in this scope

# Bing

I apologize for the confusion. The errors are due to the variables thread\_num and i not being defined in the scope of the closure that's passed to the for\_each method.

In the parallel version of the code, we need to generate the random numbers and store them in the NUMBERS array within the for\_each closure. Here's a corrected version of the generate\_random\_numbers function:

#[no\_mangle]

pub extern "C" fn generate\_random\_numbers(num\_threads: i32, num\_numbers: i32) {

let numbers\_per\_thread = num\_numbers / num\_threads;

(0..num\_threads).into\_par\_iter().for\_each(|thread\_num| {

for i in 0..numbers\_per\_thread {

let mut rand: u64 = 0;

rdrand64\_step(&mut rand);

unsafe {

\*NUMBERS.offset((thread\_num \* numbers\_per\_thread + i) as isize) = rand;

}

}

});

}

In this version, thread\_num is the variable from the into\_par\_iter method, and i is the variable from the inner for loop. Both are now defined within the scope of the for\_each closure.

Please note that this code assumes that num\_numbers is a multiple of num\_threads. If it's not, some additional logic might be needed to handle the case where the numbers cannot be evenly distributed among the threads. Also, please be aware that using raw pointers and unsafe in Rust can lead to undefined behavior if not used correctly. Always double-check your unsafe code and consider using safe abstractions whenever possible.

# User

How above loop ensures parallel processing Can you explain?

# Bing

The parallel processing in the code is achieved using the rayon crate, which provides a way to create parallel iterators in Rust.

Here's how it works:

1. (0..num\_threads).into\_par\_iter() creates a parallel iterator from the range of thread numbers. This means that the iterations of the loop can potentially be executed in parallel.
2. .for\_each(|thread\_num| {...}) applies a closure (the code inside the {...}) to each item in the parallel iterator. Because the iterator is parallel, these closures can be executed simultaneously on different threads.
3. Inside the closure, there's a loop for i in 0..numbers\_per\_thread {...} that generates random numbers. Because the outer loop is parallel, these inner loops can run concurrently on different threads.

So, the rayon crate divides the work among multiple threads, which can run on different cores of your CPU. This is how the code achieves parallel processing.

Remember, the actual degree of parallelism depends on many factors, including the number of cores in your CPU and the workload of each thread. Also, parallel processing can add complexity and overhead, so it's most beneficial when the computations are heavy and can be performed independently.