Parallel and Distributed Computing

Alberto Paoluzzi – Lecture 16 – Multi-Threading

Fri 08-04-2022

Julia High Performance: Threads

From: Sengupta, Avik: Julia High Performance: Optimizations, distributed computing, multithreading, and GPU programming with Julia 1.0 and beyond, 2nd Edition. Packt Publishing. Kindle Edition.

1. Julia High Performance: Chapter 9:
2. Julia High Performance: Threads and their life cycle:
3. Julia High Performance: The `@threads` macro:
4. Julia High Performance: Thread safety and synchronization primitives:
5. Julia High Performance: Threaded libraries:
6. Julia High Performance: Summary:

Section 1

Julia High Performance: Chapter 9:

Julia High Performance: Chapter 9:

(Concurrent Programming with) Threads

1. Threads

Julia High Performance: Chapter 9:

(Concurrent Programming with) Threads

1. Threads
2. The `@threads` macro

Julia High Performance: Chapter 9:

(Concurrent Programming with) Threads

1. Threads
2. The `@threads` macro
3. Thread safety and synchronization primitives

Julia High Performance: Chapter 9:

(Concurrent Programming with) Threads

1. Threads
2. The `@threads` macro
3. Thread safety and synchronization primitives
4. Threaded libraries

Threads: Introduction

In the last chapter, we saw how tasks provide a simple way to run multiple streams of compute, even on a single CPU core, when there is I/O involved.

1. However, as the modern CPU has grown to include multiple cores that all operate simultaneously, we need to be able to execute Julia code on them in parallel.

Threads: Introduction

In the last chapter, we saw how tasks provide a simple way to run multiple streams of compute, even on a single CPU core, when there is I/O involved.

1. However, as the modern CPU has grown to include multiple cores that all operate simultaneously, we need to be able to execute Julia code on them in parallel.
2. The way to do that is typically described as threads or threading, and, in this chapter, we talk about the facilities Julia provides to run programs on multiple CPU cores simultaneously.

Threads: Introduction

In the last chapter, we saw how tasks provide a simple way to run multiple streams of compute, even on a single CPU core, when there is I/O involved.

1. However, as the modern CPU has grown to include multiple cores that all operate simultaneously, we need to be able to execute Julia code on them in parallel.
2. The way to do that is typically described as threads or threading, and, in this chapter, we talk about the facilities Julia provides to run programs on multiple CPU cores simultaneously.
3. There are two main limitations associated with the current threading primitives that I should warn you about.

Threads: Introduction

In the last chapter, we saw how tasks provide a simple way to run multiple streams of compute, even on a single CPU core, when there is I/O involved.

1. However, as the modern CPU has grown to include multiple cores that all operate simultaneously, we need to be able to execute Julia code on them in parallel.
2. The way to do that is typically described as threads or threading, and, in this chapter, we talk about the facilities Julia provides to run programs on multiple CPU cores simultaneously.
3. There are two main limitations associated with the current threading primitives that I should warn you about.
   a. most I/Os in Julia's standard library are not thread-safe; you should not perform I/O operations in multithreaded code.

Threads: Introduction

In the last chapter, we saw how tasks provide a simple way to run multiple streams of compute, even on a single CPU core, when there is I/O involved.

1. However, as the modern CPU has grown to include multiple cores that all operate simultaneously, we need to be able to execute Julia code on them in parallel.
2. The way to do that is typically described as threads or threading, and, in this chapter, we talk about the facilities Julia provides to run programs on multiple CPU cores simultaneously.
3. There are two main limitations associated with the current threading primitives that I should warn you about.
   a. most I/Os in Julia's standard library are not thread-safe; you should not perform I/O operations in multithreaded code.
   b. The other is that nested threading is not supported very well.

Threads: Introduction

In the last chapter, we saw how tasks provide a simple way to run multiple streams of compute, even on a single CPU core, when there is I/O involved.

1. However, as the modern CPU has grown to include multiple cores that all operate simultaneously, we need to be able to execute Julia code on them in parallel.
2. The way to do that is typically described as threads or threading, and, in this chapter, we talk about the facilities Julia provides to run programs on multiple CPU cores simultaneously.
3. There are two main limitations associated with the current threading primitives that I should warn you about.
   a. most I/Os in Julia's standard library are not thread-safe; you should not perform I/O operations in multithreaded code.
   b. The other is that nested threading is not supported very well.
4. However, as you continue your journey as a Julia developer, be on the lookout for these to be fixed in future versions.

Threads: Introduction

In the last chapter, we saw how tasks provide a simple way to run multiple streams of compute, even on a single CPU core, when there is I/O involved.

1. However, as the modern CPU has grown to include multiple cores that all operate simultaneously, we need to be able to execute Julia code on them in parallel.
2. The way to do that is typically described as threads or threading, and, in this chapter, we talk about the facilities Julia provides to run programs on multiple CPU cores simultaneously.
3. There are two main limitations associated with the current threading primitives that I should warn you about.
   a. most I/Os in Julia's standard library are not thread-safe; you should not perform I/O operations in multithreaded code.
   b. The other is that nested threading is not supported very well.
4. However, as you continue your journey as a Julia developer, be on the lookout for these to be fixed in future versions.
5. So, in this chapter, we show how to write multithreaded Julia programs in ways that guarantee both safety and high performance.

Section 2

Julia High Performance: Threads and their life cycle:

Julia High Performance: Threads and their life cycle:

Section 3

Julia High Performance: The @threads macro:

Julia High Performance: The @threads macro:

Section 4

Julia High Performance: Thread safety and synchronization primitives:

Julia High Performance: Thread safety and synchronization primitives:

Section 5

Julia High Performance: Threaded libraries:

Julia High Performance: Threaded libraries:

Section 6

Julia High Performance: Summary:

Julia High Performance: Summary:

In this chapter, we saw that Julia provides a simple @thread abstraction to write multithreaded code.

1. However, there needs to be careful consideration when writing code with threads.

Julia High Performance: Summary:

In this chapter, we saw that Julia provides a simple @thread abstraction to write multithreaded code.

1. However, there needs to be careful consideration when writing code with threads.
    a. Fyou need to ensure that the number of threads you use is commensurate with the number of CPU cores you have.

Julia High Performance: Summary:

In this chapter, we saw that Julia provides a simple @thread abstraction to write multithreaded code.

1. However, there needs to be careful consideration when writing code with threads.
   a. Fyou need to ensure that the number of threads you use is commensurate with the number of CPU cores you have.
   b. you need to be careful in accessing global state.

Julia High Performance: Summary:

In this chapter, we saw that Julia provides a simple @thread abstraction to write multithreaded code.

1. However, there needs to be careful consideration when writing code with threads.
   a. Fyou need to ensure that the number of threads you use is commensurate with the number of CPU cores you have.
   b. you need to be careful in accessing global state.
   c. you need to know of any other threaded libraries within your environment.

Julia High Performance: Summary:

In this chapter, we saw that Julia provides a simple @thread abstraction to write multithreaded code.

1. However, there needs to be careful consideration when writing code with threads.
    a. Fyou need to ensure that the number of threads you use is commensurate with the number of CPU cores you have.
    b. you need to be careful in accessing global state.
    c. you need to know of any other threaded libraries within your environment.
2. With those caveats, however, you can achieve impressive performance gains when using threads with Julia.

Multi-Threading: Julia manual

Multi-Threading

Starting Julia with multiple threads

1. Data-race freedom

Multi-Threading: Julia manual

Multi-Threading

Starting Julia with multiple threads

1. Data-race freedom
2. The `@threads` Macro

Multi-Threading: Julia manual

Multi-Threading

Starting Julia with multiple threads

1. Data-race freedom
2. The `@threads` Macro
3. Atomic Operations

Multi-Threading: Julia manual

Multi-Threading

Starting Julia with multiple threads

1. Data-race freedom
2. The `@threads` Macro
3. Atomic Operations
4. Per-field atomics

Multi-Threading: Julia manual

Multi-Threading

Starting Julia with multiple threads

1. Data-race freedom
2. The `@threads` Macro
3. Atomic Operations
4. Per-field atomics
5. Side effects and mutable function arguments

Multi-Threading: Julia manual

Multi-Threading

Starting Julia with multiple threads

1. Data-race freedom
2. The `@threads` Macro
3. Atomic Operations
4. Per-field atomics
5. Side effects and mutable function arguments
6. `@thread` call

Multi-Threading: Julia manual

Multi-Threading

Starting Julia with multiple threads

1. Data-race freedom
2. The `@threads` Macro
3. Atomic Operations
4. Per-field atomics
5. Side effects and mutable function arguments
6. `@thread` call
7. Caveats

Multi-Threading: Julia manual

Multi-Threading

Starting Julia with multiple threads

1. Data-race freedom
2. The `@threads` Macro
3. Atomic Operations
4. Per-field atomics
5. Side effects and mutable function arguments
6. `@thread` call
7. Caveats
8. Safe use of Finalizers