Unity Burst

2023년 2월 9일 목요일 오후 7:29

2023.02.09 [학습 데이터]

Unity Burst는 Il2cpp와 함께 항상 거론되는 컴파일러다.

DOTS가 실 적용된 사례: 게임 [Diplomacy is not an option]

DOTS 3대장 [**C# Job system, Unity Burst, Entity**]
DOTS를 사용하려면 세가지를 모두 써야하는게 아니고, 각각 사용으로도 충분한 효과를 낼 수 있다. 다만 세가지가 같이 사용되면 더욱 큰 시너지를 낸다.

Entity: 유니티 내부를 이 엔티티를 사용해서 개조하는 방향으로 가고있다. 다만 아직은 Preview 형태로 제공중이며, 사용자들에게 강요를 하고있지않다.

C# Job system: 쓰레드를 유니티에서 안전하게 사용할 수 있는 시스템. 멀티 쓰레드 환경 구성을 보다 쉽게 만들어준다.

Unity Burst: IL*을 어셈블리어로 바꿔줌*. 단순히 언어 변환 뿐이아니라, 코드 최적화를 같이 진행한다. 강점으로는 SIMD(Single Instruction Multiple Data)에 최적화 된 코드를 만들 수 있다. 간단히 말하면 하나의 연산으로 여러 데이터를 처리할 수 있는 아키텍쳐로, ARM엔 NEON, Intel에는 SSE 같은 것이 있다. 데이터 기반의 코드로(DOD) 작성하게 되면, 이 Burst가 최적화를 잘 해준다고 한다*.

IL2CPP: C#을 IL로 바꾸고, IL(.Net Assembly 사실상 중간언어)이 다시 C++로, C++은 컴파일 과정에서 Assembly어로, 다시 기계어로 변환하는 과정.

* 1) 사실 Burst Compiler도 IL을 곧장 어셈블리어로 바꾸지 않는다. LLVM*(Low Level Virtual Machine)을 통해 IR*로 변경 후, 어셈블리어로 변경한다. 이 과정에서 IR을 통해 최적화를 해서 다시 IR을 만들고.. 몇 번의 최적화 반복 후, 어셈블리어를 생성한다.
 * 2) IL: Intermediate Language (중간 언어)
 * 3) IR: Intermediate Representation (중간 표현 ?)
 * 4) LLVM: Low Level Virtual Machine으로, 멀티 플랫폼을 타겟으로 해서 가상 머신을 통해 동작하도록 되어있다.
 * 5) Burst가 최적화를 완전히 자동적으로 해주는 건 아니고, 사용자가 최적화가 가능하도록 코드를 작성해주는 것이 필요하다. 이에 대한 내용은 후술하겠다.

Burst 최적화 코드를 짜려면?

- **컨테이너 변경:** 기존의 C#의 컨테이너들 대신, NativeArray, NativeList 같은 컨테이너들을 사용해주어야 한다. 다만 이런 경우는 사용이 끝난 경우 직접 Dispose해주어야 하며, 이는 메모리 관리를 직접 해줘야한다는 뜻이다. Native~ 는 Burst Compiler와 Job System에 최적화된 컨테이너로써, 멀티 쓰레드 환경에서 일반적인 배열과 리스트보다 훨씬 빠른 성능을 보여주며, 쓰레드 세이프 하다.

- **Struct 사용:** Class 대신 Struct를 사용해야 Burst의 이점을 살릴 수 있다. Class는 레퍼런스를 받아 같은 곳을 참조하는 반면 Struct는 Value 값을 복사하여 사용한다. 멀티 쓰레드 환경에서는 데이터의 여러 복사본이 필요하며, 이 때문에 Struct를 사용하는 경우 효율적으로 동작한다.

- **가상 함수 사용 피하기:** 가상 함수는 추가적인 연산을 유발하여 코드 속도가 느려질 수 있습니다.

- **수동 메모리 관리 사용:** Burst 컴파일러는 수동적으로 메모리 할당 및 해제를 관리하는 코드에서 잘 작동합니다. 자동 메모리 관리 시스템(예: 가비지 컬렉션)을 사용하지 않는 것이 좋습니다.

- **분기 최소화:** 코드에서의 분기는 성능 문제를 야기할 수 있으며, Burst 컴파일러는 사전에 어떤 분기가 취해질지 판단하기 어려울 수 있습니다. 가능한 한 코드에서의 분기를 최소화하세요.

- **동적 배열을 사용 피하기:** 동적 배열은 자주 크기 조절이나 메모리 할당이 필요하기 때문에 Burst 컴파일러에서 최적화하기 어렵습니다. 고정 크기 배열이나 다른 데이터 구조를 사용하는 것이 좋습니다.

- **간단한 데이터 구조 사용:** Burst 컴파일러는 배열, 구조체, 값 타입과 같은 간단한 데이터 구조에서 최상의 성능을 발휘합니다. 연결리스트, 트리, 그래프와 같은 복잡한 데이터 구조는 성능 문제가 발생할 수 있습니다.

- **암시적 변환을 사용하지 마십시오:** 암시적 변환은 추가적인 처리가 필요하기 때문에 성능 문제를 일으킬 수 있습니다. 명시적 변환을 사용하는 것이 좋습니다.

- **코드를 정기적으로 프로파일링하기:** 코드가 Burst 컴파일러에 최적화되도록 유지하기 위해서 정기적으로 코드를 프로파일링하는 것이 필수입니다. 이를 통해 성능의 병목 지점을 식별하고 개선할 수 있습니다.

[Burst로 컴파일 하기]

1. **Burst 패키지 설치:** Unity를 사용하고 있다면 Unity Package Manager를 사용하여 Burst compiler를 설치할 수 있습니다. Unity Package Manager 창에서 Burst 패키지를 검색하세요.
2. **코드 주석 추가:** Burst에 어떤 메소드를 Burst compiler로 컴파일할지 알려주어야 합니다. 이를 위해서 [BurstCompile] 어트리뷰트를 메소드에 추가해야 합니다.
3. **Burst로 컴파일:** 코드에 어트리뷰트를 추가한 후 프로젝트를 빌드하세요. Burst는 어트리뷰트가 추가된 메소드를 자동으로 감지하여 고급 최적화 기술을 사용하여 컴파일합니다.