SwarmStorm: Coarrays in Satellite Simulations

Daniel T. and Achates

December 1, 2024

Abstract

This document explores the application of Fortran coarrays to simulate satellite engagements. By leveraging coarrays' parallelism and synchronization capabilities, the simulation models packs of satellites as coarray images and individual satellites as coarray indices. The report outlines the foundational principles, architectural considerations, and implementation details of this innovative approach.

1 Genesis

The use of coarrays in satellite simulations provides a powerful paradigm for modeling distributed systems. This section explores the foundational concepts and initial design considerations for simulating satellite engagements using coarrays in Fortran.

1.1 Concepts and Design Philosophy

The simulation models satellite engagements, where:

- **Packs of Satellites**: Each pack is represented by an image in the coarray model.
- **Individual Satellites**: Satellites within a pack are indexed as elements in coarrays.
- **Engagement Dynamics**: Packs engage enemy packs, sharing data and state across images via coarray communication.

This structure leverages Fortran's coarray capabilities to parallelize and simplify the simulation:

- **Parallelism**: Each image independently simulates its pack, enabling concurrent computation.
- **Data Sharing**: Coarray variables facilitate efficient inter-image communication.
- **Scalability**: The design can accommodate increasing numbers of satellites and packs.

1.2 Initial Considerations

The key architectural choices are:

- **Packs as Images**: Each image represents a pack of satellites. Internal dynamics are simulated locally, while inter-pack engagements are handled via coarray references.
- **Satellites as Coarray Indices**: Satellites within a pack are indexed in coarrays. This simplifies data management and ensures efficient access to satellite states.
- **Synchronization**: Use coarray synchronization primitives (`sync all`, `sync team`, etc.) to manage interactions between packs.
- **Encapsulation**: Encapsulate satellite behavior within types, ensuring clean separation between simulation logic and data structure.

1.3 High-Level Design

The simulation follows this high-level flow:

1. **Initialization**: Distribute satellite data across images and initialize states (e.g., position, velocity, fuel).
2. **Engagement Simulation**: Simulate pack dynamics within each image and coordinate inter-pack engagements using coarray communication.
3. **Result Aggregation**: Gather and summarize results for analysis (e.g., pack status, satellite losses).

This design ensures scalability, modularity, and efficient use of Fortran's coarray features. Future sections will explore specific implementation details and results of these simulations.

2 Causal Bubbles

The concept of **causal bubbles** offers a powerful abstraction for reducing interdependencies and enhancing the efficiency of parallel discrete event simulation (PDES) in satellite engagements. This section outlines the principles, benefits, and implementation strategies for leveraging causal bubbles with coarrays.

2.1 Definition and Advantages

A causal bubble is an isolated group of events or processes that interact only within the bubble, without affecting or being affected by external processes. This approach has several advantages:

- **Reduced Communication Overhead:** Events in one mission bubble do not impact another, minimizing the need for inter-bubble communication.
- **Independent Processing:** Each mission can proceed independently, leveraging parallelism effectively.
- **Scalability:** The simulation can handle increasing numbers of missions without a proportional increase in complexity.

2.2 Local Timescales

Each mission operates on a local timescale, which can have finer granularity than the global simulation clock. This allows missions with faster dynamics to resolve events more accurately without slowing down the overall simulation. For example:

- The global simulation clock advances in 1-second intervals.
- Missions resolve their events at a finer resolution, such as 0.1 seconds.

2.3 Implementation Strategy

2.3.1 Using Coarrays

Coarrays provide a natural way to map missions and their satellites to parallel processes:

- Each **mission bubble** is assigned to a coarray image or a team of images.
- **Synchronization** within a bubble is managed using `sync team`, while `sync all` is used only for global reporting or major events.

2.3.2 Example Code

The following illustrates the implementation of local timescales and event resolution:

```
real :: global_time = 0.0   ! Overall simulation clock
real :: mission_time = 0.0  ! Local clock for the mission

do while (global_time < max_simulation_time)
    ! Local events
    mission_time = mission_time + 0.1
    call resolve_local_events(mission_time)

    ! Synchronize with global clock periodically
    if (mod(global_time, 1.0) == 0.0) then
        sync team
    end if

    global_time = global_time + 1.0
end do
```

2.4 Benefits of the Approach

By treating missions as causal bubbles:

- **Parallelism is Maximized:** Independent missions can run simultaneously on separate images.
- **Accuracy is Preserved:** Local timescales allow finer resolution where needed.
- **Synchronization Overhead is Minimized:** Inter-bubble communication is avoided unless absolutely necessary.

This design ensures an efficient and scalable simulation framework for modeling satellite engagements.