5/16/2024

Parallel and Distributive Computing

CEN 455

PCAM METHODOLOGY

SUBMITTED TO

MA'AM SAIMA

Made by:

Hannaan Ahmed Jadran
01-134211-106

Muhammad Ali Momin
01-134211-051

Table of Contents

Problem Statement

By applying the PCAM methodology to the procedural generation algorithm, Graphics module, and Movement module in Unity, we can design an efficient parallel solution. Partitioning the tasks, using appropriate communication strategies, agglomerating tasks to balance the load, and employing suitable mapping techniques ensure optimal performance and scalability. This approach leverages parallel computing to handle complex room generation, rendering, and movement tasks efficiently, enhancing the overall effectiveness of the system.

Methodology

Partitioning Techniques

To divide the problem into smaller tasks that can be executed concurrently, we utilize domain decomposition. This involves breaking down the grid into smaller, manageable sections.

- **Spatial Decomposition:**

  The entire grid is divided into multiple sub-grids. Each sub-grid represents a distinct portion of the map where room generation and pathfinding will occur independently.

  Example: If the main grid is 16x16, we can partition it into 4x4 sub-grids, resulting in 16 smaller sub-grids. Each sub-grid is a separate domain that can be processed independently.

- **Task Decomposition:**

  Within each sub-grid, tasks involve generating rooms, ensuring connectivity, and managing paths using the DFS algorithm. Each task corresponds to operations on the rooms within a particular sub-grid.

  Example: Each processor is responsible for generating rooms in its assigned sub-grid, handling DFS to find paths within the sub-grid, and backtracking when necessary.

  By dividing the grid spatially and assigning each sub-grid to a processor, we reduce dependencies and enable parallel processing. Each processor works on its sub-grid independently, which minimizes contention and maximizes parallel efficiency.

  Techniques: Unity.Jobs, Burst Compiler, Parallel.ForEach, Custom Threading

- ### Graphics Module

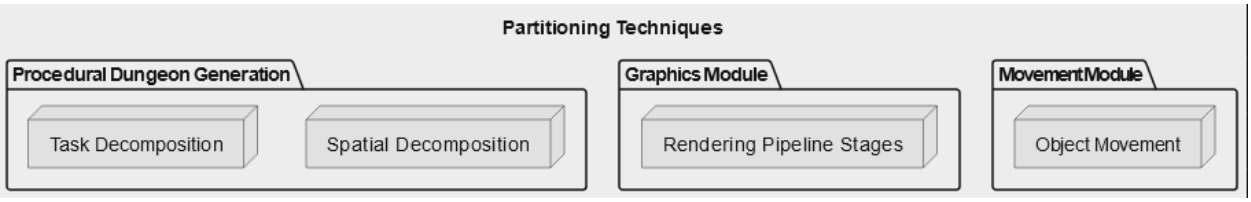
  Unity's Graphics module can be partitioned based on the rendering pipeline stages, such as geometry processing, vertex shading, rasterization, and pixel shading. These stages can be executed in parallel on the GPU, leveraging its massively parallel architecture.

  Techniques: Compute Shaders

- ### Movement Module

  The Movement module can be partitioned based on the objects or entities that need to be moved. Each object's movement calculations can be handled as a separate task, which can be executed in parallel on the CPU.

  Techniques: Unity.Jobs, Burst Compiler, Parallel.ForEach, Custom Threading



Communication Requirements

- ### Message Passing

  Communication between processors is essential to ensure the generated rooms form a coherent, connected structure across sub-grids. We use message passing to facilitate this communication.

- ### Boundary Synchronization

  Processors handling adjacent sub-grids must exchange information about room placements at the boundaries to ensure seamless transitions.

  Example: If a room in sub-grid A is at the boundary with sub-grid B, processors responsible for A and B must coordinate to ensure the doorways align and the paths connect.

- ### State Sharing

  Processors need to share their state when performing operations like backtracking or when they reach the boundaries of their sub-grids.

Example: When a processor reaches a boundary and needs to continue the path in an adjacent sub-grid, it communicates the current state (e.g., current room, visited nodes) to the processor handling the adjacent sub-grid.

Message passing is efficient for our needs as it minimizes the overhead of communication by only requiring processors to exchange information at specific points, such as boundaries, rather than continuously.

Techniques: Shared data structures (NativeArrays, NativeQueues), thread-safe collections, shared memory, locks, semaphores, message queues

- ## Graphics Module

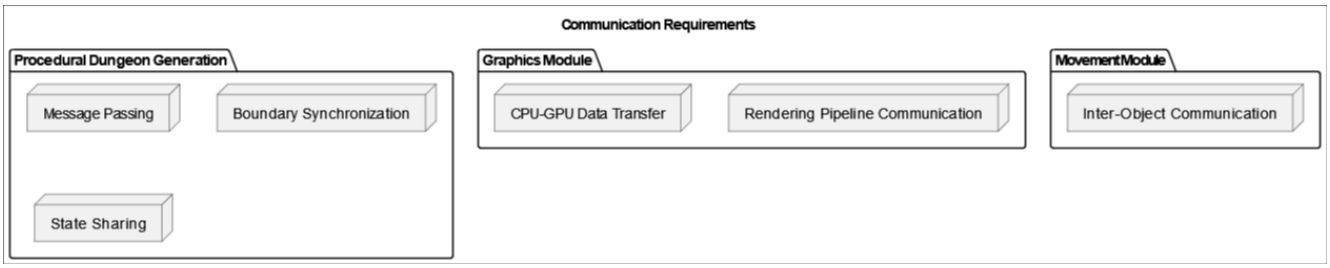
  In the Graphics module, communication may be required between different stages of the rendering pipeline, as well as between the CPU and GPU for scene data transfer.

  Techniques: Shared textures, buffer resources, asynchronous data transfers, synchronization points

- ## Movement Module

  The Movement module may require communication between tasks for collision detection, path-finding, or other interactions between objects.

  Techniques: Shared data structures, thread-safe collections, shared memory, locks, semaphores, message queues



Agglomeration Strategy

After partitioning the tasks, we need to combine smaller tasks to balance the load across processors and reduce communication overhead.

- ### Task Merging

  Smaller room generation tasks within a sub-grid are merged into larger tasks to minimize the number of inter-processor communications.

  Example: Instead of each processor handling individual rooms, it handles the entire room generation process within its assigned sub-grid, thus reducing the frequency of communications.

- ### Load Balancing

  Adjust the size of sub-grids dynamically based on the complexity of room generation to ensure each processor has a balanced workload.

  Example: If some sub-grids are more complex due to a higher number of rooms or intricate paths, they can be divided further or assigned more powerful processors to ensure even workload distribution.

  Combining tasks and ensuring balanced workloads across processors helps in minimizing idle times and communication needs, leading to more efficient execution.

  Techniques: Job merging/splitting, adjusting parallel loop grain size, thread affinity

- ### Graphics Module

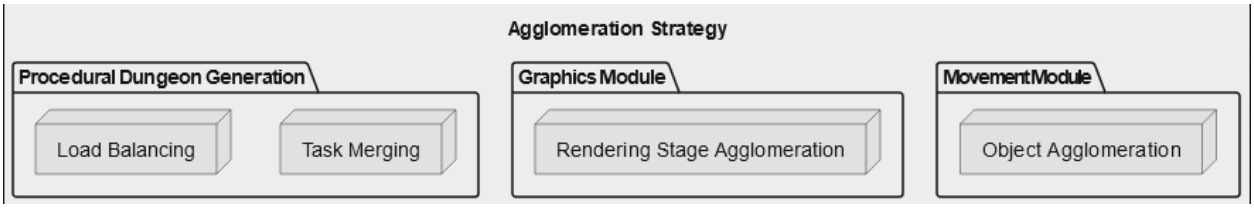
  In the Graphics module, certain rendering stages can be agglomerated to reduce data transfers between the CPU and GPU, or to leverage specific GPU hardware features more efficiently.

  Techniques: Techniques: Combining compute shaders, combining asynchronous operations

- ### Movement Module

  In the Movement module, objects that are spatially close or have similar movement patterns can be agglomerated for better cache utilization and reduced communication overhead.

  Techniques: Job merging/splitting, adjusting parallel loop grain size, thread affinity



Mapping

Mapping involves assigning tasks to processors in a way that maximizes parallel efficiency. For our grid structure, a static mapping approach is appropriate.

- ## Processor Assignment

  Assign each sub-grid to a specific processor. Each processor is responsible for generating rooms and managing paths within its sub-grid.

  Example: In a 16x16 grid divided into 4x4 sub-grids, each of the 16 processors handles one sub-grid. Processor P1 handles sub-grid 1, Processor P2 handles sub-grid 2, and so on.

- ## Concurrency Degree

  The degree of concurrency is determined by the number of processors and the size of sub-grids. With 16 processors, we achieve a concurrency degree of 16, meaning 16 sub-grids are processed in parallel.

  Static mapping ensures that each processor consistently works on a specific sub-grid, reducing the overhead associated with dynamic task allocation and simplifying communication. This approach also allows for straightforward load balancing and task management.

  Techniques: Static mapping, Unity Job System, thread affinity

- ## Graphics Module

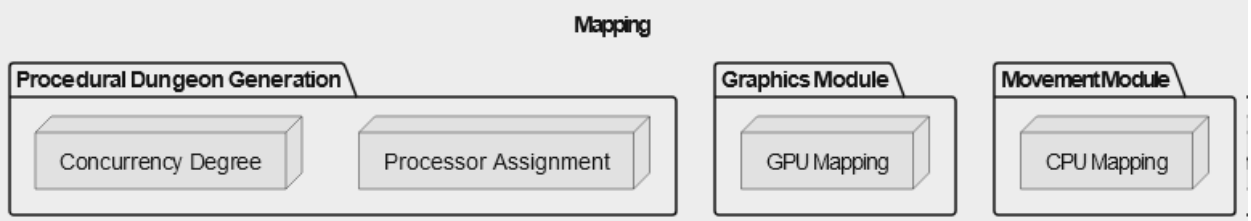
  In the Graphics module, different rendering stages can be mapped to the GPU's parallel processing units (e.g., CUDA cores, compute units) based on their parallelism and resource requirements.

  Techniques: GPU mapping, compute shader mapping

- ## Movement Module

For the Movement module, tasks can be mapped to CPU threads or cores based on the number of objects and their movement complexity.

Techniques: CPU mapping, Unity Job System, thread affinity



Conclusion

By applying the PCAM methodology to the procedural generation algorithm, Graphics module, and Movement module in Unity, we can design an efficient parallel solution. Partitioning the tasks, using appropriate communication strategies, agglomerating tasks to balance the load, and employing suitable mapping techniques ensure optimal performance and scalability. This approach leverages parallel computing to handle complex room generation, rendering, and movement tasks efficiently, enhancing the overall effectiveness of the system.