FUTURES-AMR: Towards an Adaptive Mesh Refinement Framework for Geosimulations

Ashwin Shashidharan
Department of Computer Science, North Carolina State University, Raleigh, USA
ashdharan@ncsu.edu

Ranga Raju Vatsavai
Department of Computer Science, North Carolina State University, Raleigh, USA
rrvatsav@ncsu.edu

Derek B. Van Berkel
Center for Geospatial Analytics, North Carolina State University, Raleigh, USA
dbvanber@ncsu.edu

Ross K. Meentemeyer
Center for Geospatial Analytics, North Carolina State University, Raleigh, USA
rkmeente@ncsu.edu

Abstract

Adaptive Mesh Refinement (AMR) is a computational technique used to reduce the amount of computation and memory required in scientific simulations. Geosimulations are scientific simulations using geographic data, routinely used to predict outcomes of urbanization in urban studies. However, the lack of support for AMR techniques with geosimulations limits exploring prediction outcomes at multiple resolutions. In this paper, we propose an adaptive mesh refinement framework FUTURES-AMR, based on static user-defined policies to enable multi-resolution geosimulations. We develop a prototype for the cellular automaton based urban growth simulation FUTURES by exploiting static and dynamic mesh refinement techniques in conjunction with the Patch Growing Algorithm (PGA). While, the static refinement technique supports a statically defined fixed resolution mesh simulation at a location, the dynamic refinement technique supports dynamically refining the resolution based on simulation outcomes at runtime. Further, we develop two approaches - asynchronous AMR and synchronous AMR, suitable for parallel execution in a distributed computing environment with varying support for solution integration of the multi-resolution results. Finally, using the FUTURES-AMR framework with different policies in an urban study, we demonstrate reduced execution time, and low memory overhead for a multi-resolution simulation.

2012 ACM Subject Classification Computing methodologies → Distributed simulation, Computing methodologies → Multiscale systems, Applied computing → Environmental sciences

Keywords and phrases Adaptive mesh refinement, Geosimulation, Distributed system, Multi-resolution, Urban geography

Digital Object Identifier 10.4230/LIPIcs.GIScience.2018.16

1 Introduction

Over the past decade, advancements in remote sensing technologies and classification techniques have increased the availability of high-resolution datasets relevant to urban simulation. High resolution LiDAR derived DEMs, land cover classifications, and the increasing amount of vector-based spatial layers promise to deliver a better understanding of urbanization for forecasting urban development. However, in practice, computational constraints impact

the resolution of input data, or the extent of the study region used in an urban simulation. Particularly, memory and I/O constraints limit studies leveraging high-resolution data to small study extents, while study of large extents are often only possible with low-resolution data. Although an urban simulation may require high-resolution data only in a small region of the study (as shown in Fig. 1b), current urban simulation frameworks do not support selectively varying the resolution of a simulation for different regions (as shown in Fig. 1c and Fig. 1d) at runtime. Further, if new urbanization is highly likely only on a small portion of the study extent, modifying the urban growth simulation to use high-resolution data over the complete study extent is highly inefficient.

Adaptive mesh refinement (AMR) is a technique that can support multi-resolution simulations using high-resolution data in regions where it is necessary. For urban growth simulations in large study extents, adaptive mesh refinement at runtime would allow focusing computational resources for simulating emerging urban patterns in regions of interest (ROIs). An AMR approach using high-resolution data would account for more prominent local effects like topographic features and land cover classes to simulate accurate urbanization patterns. Additionally, using low-resolution data for simulation in regions of less importance would reduce memory overhead and enable faster simulation. In effect, such an approach would eliminate the computational overhead of a high-resolution simulation over the global extent of a study region, while generating fine spatial patterns where necessary.

Although an AMR approach promises significant computational savings, AMR techniques developed thus far only support refinement and coarsening criteria for solving partial differential equations (PDEs) in a scientific simulation. In particular, these are not applicable to geosimulations which use cellular automaton (CA) based models to generate urbanization outcomes. Thus, the first challenge is the development of new refinement and coarsening criteria to support AMR with geosimulations like urban growth. In particular, geosimulations require a *mesh placement strategy* that specifies the location, extent and spacing of a mesh (resolution), and a *mesh generation strategy* for use with different datatypes in the simulation. In turn, the choice of a mesh generation strategy impacts the integration strategy for synchronizing the results generated at different resolutions.

In this paper, we address this research gap and develop a distributed AMR framework, FUTURES-AMR that supports multi-resolution geosimulations. Specifically, the framework supports *refinement* and *coarsening* requests using multi-resolution data in regions of interest (ROIs) for two scenarios: (i) *static refinement* in ROIs specified by an end user; (ii) *dynamic refinement* based on a combination of static policies and the simulation outcomes at runtime. For both scenarios, we allow end users to specify static policies that define refinement and coarsening criteria for the AMR simulation. Finally, we develop two approaches - asynchronous AMR and synchronous AMR with different load balancing and solution integration strategies in a master-worker style distributed system architecture.

The rest of the paper is organized as follows: in Sect. 2, we summarize existing research for AMR simulation. In Sect. 3, we provide an overview of Adaptive Mesh Refinement as used in numerical analysis. In Sect. 4, we describe our distributed system architecture for AMR in the asynchronous and synchronous AMR approaches, and how we adapt the FUTURES geosimulation in our AMR framework. In Sect. 5, we describe our experimental setup and present results from executing FUTURES-AMR in two different geographic regions with user-defined policies. Finally, we conclude in Sect. 6, with future work in Sect. 7.

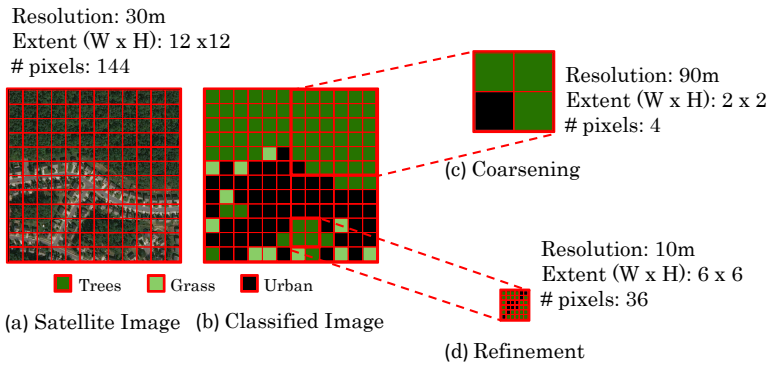


Figure 1 Illustration of the proposed FUTURES-AMR framework – The study extent shown in the classified image has 60% non-urban pixels. Refinement using 10m resolution data, and coarsening using 90m resolution data are requested on 5% and 40% of the total non-urban pixels in the study area, respectively. The default simulation at 30m resolution executes on the remaining 55% of non-urban pixels.

2 Related Work

Adaptive mesh refinement (AMR) is a technique that can be used with both structured and unstructured meshes. AMR techniques support dynamically adjusting the cell spacing on a mesh to achieve an accurate numerical solution. Structured adaptive mesh refinement (SAMR) was first proposed by Berger et al. [4] to solve partial differential equations (PDE) in shock hydrodynamics. This technique which relies on partitioning the problem space into different regions with varying spatial resolutions is achieved by imposing varying resolution grids in space. Further, each region is assumed to be rectangular in shape with a grid hierarchy to represent the relationships between different regions. As the solution progresses, nested grids or new grids are generated refining the problem in these regions. In case of time-dependent equations, these refinements can be applied to compute solutions at finer temporal resolutions as well.

Initially developed to solve simulations using hyperbolic conservation laws [4, 3], AMR approaches have since been extended to solve parabolic and elliptic equations. These numerical solvers find widespread use across various domains such as Computational Fluid Dynamics [4, 3], Astrophysics [9] and Climate Modeling [18]. General-purpose AMR frameworks have also been developed that support developing applications not specifically designed for a domain. BoxLib [2], Chombo [6] and SAMRAI [24] are examples of such frameworks with numerical solvers and APIs for developing codes for new applications. A comprehensive listing of the different frameworks for AMR refinement and their applications can be found in a survey by Dubey et al. [8].

AMR frameworks typically define a grid hierarchy management scheme to handle the coarse and fine regions. Block representation schemes have been devised which represent regions as grids using lower and upper coordinates of a bounding box [4, 2, 6, 24]. Similarly, tree representations exist that define coarse and fine regions in terms of parent-child relations and their splitting criteria [10, 20, 12]. These representations have implications on the number of cells to refine, the data distribution strategy for parallel computation, memory requirements and storage overhead. Exclusive computational geometry libraries also exist which support creation of structured and unstructured meshes for scientific simulations. CGAL [15], Silo [19], PARAMESH [12] are examples of libraries with geometry algorithms

and mesh generation and management routines for use in scientific applications. However, these libraries lack support for numerical solvers and AMR refinement operations. PETSc [1] and Hypre [14] are libraries with parallel numerical solvers. Even so, combining individual libraries to port existing serial code and develop a parallel AMR application requires parallel programming expertise and significant rework.

Owing to the numerical complexity of solving partial differential equations (PDEs), most AMR related research has focused on developing data structures and algorithms to support parallel and distributed computation [7]. These frameworks are designed with one of the two popular load balancing strategies for AMR: *patch-based* and *domain-based* (or tree-based). In the patch-based approach [4, 23], load balancing distributes regions for refinement over a set of processors using a binning, greedy or round-robin technique. Although, a patch-based approach offers a simple load balancing strategy to balance overall computational work at a processor, data movement for synchronizing results across refinement levels is unavoidable and could lead to significant communication overhead. On the other hand, domain-based approaches attempt to optimize communication overhead by assigning coarsening or refinement operations for a sub-region to a processor where its parent region resides [10, 20, 12]. However, domain-based approaches suffer from scalability issues at higher levels of refinement as dynamic reconfiguration of the workload necessitates data migration to maintain the load and avoid synchronization between nested levels at each processor. A comprehensive comparison of the parallelization techniques for dynamic load balancing can be found in a survey by Rantakokko et al. [21]. The results of the survey indicate that no single partitioning scheme performs best across all types of applications and systems. Finally, AMR frameworks typically also define techniques to integrate results at the boundary of coarse-fine interfaces. Refluxing and circulation integration techniques, which combine results from interpolation of low resolution data at coarser levels and aggregation of data at finer levels are used to update PDE solutions at the boundaries.

General-purpose parallel AMR frameworks attempt to reduce the programming effort to develop parallel structured AMR applications. While most of these frameworks are distributed memory implementations [9, 6, 24], AMRCLAW [5] is a shared memory implementation. Parallel AMR frameworks facilitate development of parallel AMR applications by handling data organization and distribution, load balancing and data communication as part of the framework [17]. Along with numerical solvers for PDEs, these frameworks abstract the implementation details such as the data type, parallel communication patterns and data placement strategies from the user. AMR frameworks [16, 11] also exist that compute solutions in irregularly shaped regions of the sub-domain without assuming a logically rectangular structure. However, similar to structured AMR frameworks these are only suitable for scientific applications using partial differential equations. Finally, we are not aware of AMR frameworks developed to support geosimulations.

3 Adaptive Mesh Refinement

To compute a numerical solution for PDEs, an adaptive mesh refinement technique starts by imposing a coarse grid (or mesh) over the complete problem domain. The grid defines the cell spacing, or resolution for computation in the domain. Imposing a finer grid in the domain introduces more grid points while, a coarse grid presents fewer points at which, solutions for the equation must be calculated. Thus, the computational complexity to solve a PDE depends on the grid spacing of the domain. An adaptive mesh refinement technique superimposes fine grids only in certain sub-domains (or regions) of the problem (also known

as regridding). These are identified by estimating the accuracy or error of the computed solution. Finer grids are recursively imposed in the region till the error or accuracy of the computed solution is acceptable (i.e., below or above a threshold), or a maximum level of refinement is reached. Specifying a maximum level of refinement avoids infinite recursion in the regridding step. Thus, in an AMR based solution, a coarse grid is applied on the complete problem domain, but recursively refined in regions till a suitably accurate solution is obtained.

In regions superimposed with finer grids, AMR uses interpolation to resolve the initial values at the fine grid points from the coarse grid points. Subsequently, the solutions of the equations at the finer grids points are computed, and results at the fine grid points are aggregated to update the solution at the coarse grid points. Along the fine-coarse grain region boundaries, the AMR integration approach uses a flux conservation or circular integral control technique to update values at the coarse grid points. Thus, a solution at the initial coarse resolution (or default resolution) for the complete domain is obtained using AMR.

4 FUTURES-AMR

In our framework, we modify the Berger-Oliger-Collela approach [4] to support adaptive mesh refinement for a geosimulation. We make two major modifications in the four step Berger-Oliger-Collela approach. Firstly, we substitute the problem of solving PDEs at different intervals in a domain with an urban growth simulation using a Patch Growing Algorithm (PGA) [13] in a geographic region. Secondly, we modify the error-based AMR refinement criteria for PDE solvers with AMR refinement criteria based on user-defined policies for a region. Thus, in our FUTURES-AMR framework, the FUTURES urban simulation executes the PGA at different resolutions based on refinement criteria expressed in user-defined policies.

We also make a few assumptions about supported geosimulations in this framework. First, a geosimulation executing in this framework is assumed to be a cellular automaton consisting of a grid of cells with transition rules such as defined by a PGA. Second, each cell has a fixed spatial resolution representing a fixed area on the landscape. A geosimulation begins at this fixed resolution over the complete landscape. Third, the transition rules of the CA-based geosimulation for patch growth must be specified, or generalizable for use at different resolutions. Based on these assumptions and modifications, we define the FUTURES-AMR algorithm as follows:

▶ **Step 1.** Start a geosimulation with a coarse default resolution over the complete study extent.

▶ **Step 2.** Evaluate static policies as part of PGA to identify regions that need higher/lower resolution data.

▶ **Step 3.** Superimpose finer grids for refinement or coarser grids for coarsening in these regions. Subsequently, execute PGA till either the PGA halting criteria is met or higher resolution data is unavailable.

▶ **Step 4.** Integrate multi-resolution simulation outcomes from refinement and coarsening in different regions with the default resolution result in the global extent.

We design two simulation approaches namely, asynchronous AMR and synchronous AMR that vary in their implementation of Step 4. We describe these approaches and their varying support for policies in Section 4.3 and 4.4. We begin with a brief description of the PGA for the proposed framework.

4.1 Patch Growing Algorithm (PGA)

In the simulation of an urban landscape, new urban patches are developed by executing a Patch Growing Algorithm at suitable development sites in the landscape. One standard method [13] is to determine a suitable seed and execute a neighbor discovery process to determine new cells for urban patch growth. The PGA generates new patches that characterize the spatial changes due to urbanization in terms of *patch shape* and *patch size* starting at the seed location. However, the algorithm depends on a fine grid to capture these patterns at a fine granularity. In general, wider spacing of grid points results in lower data resolution representing the landscape and hence, higher inaccuracy in patterns of the generated patches. These solutions may be acceptable in certain regions of a landscape, e.g. in a sparsely populated remote rural region, but not in a dense urban region like a central business district (CBD). Thus, to support varying mesh spacing depending on the requirement in a region, we modify the PGA to generate refinement and coarsening requests at runtime.

4.1.1 Refinement/Coarsening

In FUTURES-AMR, a refinement or coarsening request is generated in response to user-defined policies in a region. These policies (see Section 4.2) define a refinement or coarsening criteria in a region for use during the simulation. A refinement criterion imposes a finer grid in a buffer region surrounding the seed site. In turn, the simulation executes the PGA using high-resolution data (resolution higher than the default resolution) in this region. Besides fine grids, coarse grids may also be specified for patch growth using PGA. In case of coarse grids, the simulation uses low-resolution data (resolution lower than the default resolution) in this region for the PGA. In case of both, fine and coarse grids, further refinement may be triggered to meet the PGA halting criteria until a higher resolution of data is unavailable. Thus, the simulation proceeds in discrete time-steps executing the PGA at default resolution, or by refining, or coarsening select regions in the geographic extent. The simulation result at the end of each time-step is a collection of coarsening results, refinement results and the simulation result at the default resolution. We formally define a refinement and coarsening request as follows:

$$X(L, E, r) \leftarrow P_1 \wedge P_2 \wedge \ldots \wedge P_n \tag{1}$$

where X is either a refinement or coarsening request, L is the geolocation of the request, E is the extent to refine or coarsen from L, r is the resolution to use with the request, and $P_1 \ldots P_n$ are user-defined policies in the extent E.

4.2 Policy Specification

In our AMR framework, we support user-defined static policies specified as input to the simulation. These policies serve as refinement and coarsening criteria for a simulation to perform *static* or *dynamic* refinement. If a geosimulation is unable to satisfy urbanization conditions using low-resolution data, refinement is triggered. Similarly, satisfying development conditions by coarsening with low-resolution is also supported. We formally define a policy as follows:

$$P \leftarrow A_1 \wedge A_2 \wedge \ldots \wedge A_n \tag{2}$$

where, A_i is a spatial or non-spatial attribute, and P is a user-defined policy expressed as a conjunction of such attributes.

4.2.1 Static Refinement

Static refinement is a technique used to a priori superimpose meshes in regions of interest. In these regions, the mesh resolution is adjusted once, which is then maintained throughout the simulation. In case of geosimulations, as described in Section 4.1, coarsening may also be acceptable in certain regions of the landscape. Thus, in the FUTURES-AMR framework static policies specified a priori support both, refinement and coarsening criteria in a region. Static policies specify such regions where simulations with a different resolution must be carried out. For example:

P1: A *spatial refinement policy* specifies a polygon feature and resolution of data (1m/10m) to simulate patterns of urban development.

P2: A *spatial coarsening policy* specifies a polygon feature and resolution of data (90m/270m) to simulate patterns of urban development.

4.2.2 Dynamic Refinement

Dynamic refinement is carried out in response to conditions arising during a simulation. In case of dynamic refinement, fine meshes are superimposed in regions based on a combination of simulation outcomes and a refinement criteria satisfied at runtime. The same is applicable to coarsening as well. In the FUTURES-AMR framework, refinement criteria for patch growth is defined using static policies. For example:

P1: A *patch growth refinement policy* specifies high-resolution data (1m/10m) to develop urban patches smaller than a given size within a distance from a central business district.

P2: A *patch growth coarsening policy* specifies low-resolution data (90m/270m) to develop urban patches greater than a given size beyond a distance from a central business district.

P3: A *data-driven policy* specifies the resolution of data (1m/10m/90m/270m) to develop urban patches based on site development potential determined at runtime.

In the simulation, during PGA execution, these policies are evaluated at runtime to determine if dynamic refinement is necessary. A data-driven policy (e.g., P3) serves to resolve potential conflicts in case of multiple user-defined policies. If dynamic refinement is triggered, PGA iteratively refines the mesh to simulate urbanization till the refinement criteria is met. Thus, in dynamic refinement, the PGA adaptively adheres to the structure of the patch being developed at higher resolutions.

4.3 Asynchronous AMR

The asynchronous approach in our AMR framework is designed to support experimentation of policies at different resolutions in a study area. To be able to compare outcomes due to a user-defined policy, the approach executes the simulation, both in the presence and absence of policies, and generates results at different resolutions. In particular, the approach executes the PGA at multiple resolutions only in regions with user-defined policies, avoiding the execution overhead of a multi-resolution simulation over the complete study extent. Further, the emerging spatial structures in a time-step at different resolutions are retained as-is, eliminating additional I/O required to aggregate the results generated at different resolutions.

4.3.1 Solution Integration

In the asynchronous AMR approach, the results from adaptive mesh refinement are not integrated with the solution computed at the default resolution. Such an approach preserves

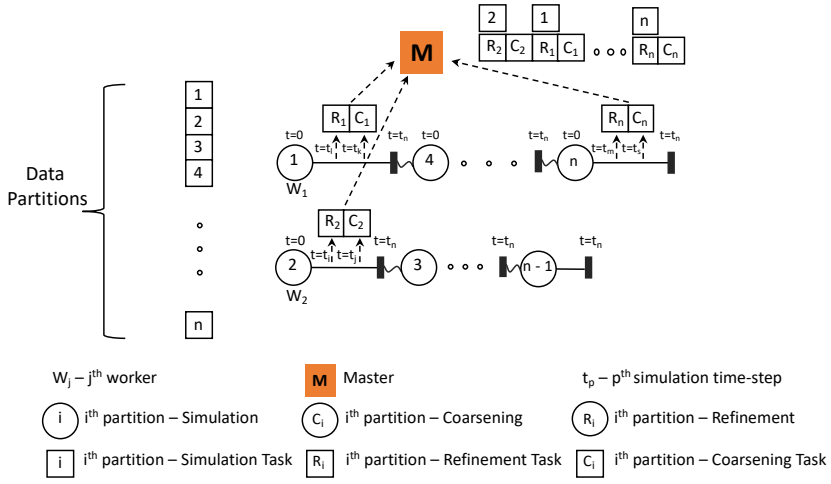


Figure 2 Asynchronous AMR (Phase 1) – Each worker executes a simulation generating coarsening and refinement requests at each time-step of the simulation. The master receives and aggregates these requests from the workers at every time-step for processing in Phase 2.

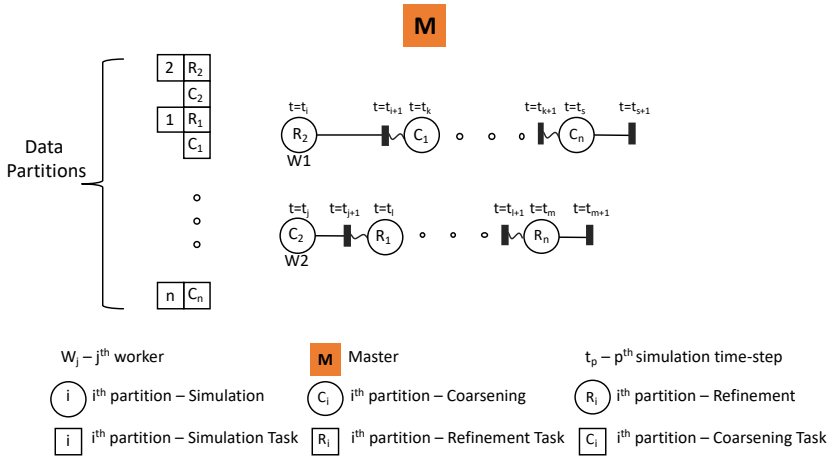


Figure 3 Asynchronous AMR (Phase 2) – The workers execute the refinement and coarsening requests from Phase 1 as assigned by the master.

the patch specificity obtained from multi-resolution simulations of different regions. GIS overlay techniques can be used to visualize the result layers from refinement and coarsening in different regions along with the global output raster.

4.3.2 Load Balancing

In the asynchronous approach, refinement and coarsening requests are processed independent of the simulation at the default resolution. Refinement and coarsening requests triggered by the simulation execute asynchronously without blocking the simulation. The approach ensures maximum resource utilization throughout the simulation.

We implement a master-worker approach for distributed asynchronous AMR simulations. In Phase 1 of this approach (Fig. 2), we begin by assigning each worker a geographic partition for simulation. Each worker executes a simulation on its partition generating new refinement

and coarsening requests. At the end of every time-step, the worker relays these requests to the master. Finally, once a worker completes all time-steps of the simulation on its assigned partition, the master schedules a new partition at the worker, if any.

Phase 2 (Fig. 3) begins when all partitions in the study have been processed. In Phase 2, the master schedules the refinement and coarsening requests received from the workers during Phase 1. Similar to Phase 1, each worker receives a refinement or coarsening request till all requests at the master have been processed. Finally, if further refinement becomes necessary while processing a request at a worker, it is executed at the same worker.

4.4 Synchronous AMR

The synchronous AMR approach propagates the effects of static policies in each time-step of the geosimulation to the subsequent time-step of the simulation. In this approach, spatial structures that emerge due to a policy at a particular time-step are input to the next time-step, i.e., the spatial effects of policies are temporally preserved as well. Specifically, the simulation outcomes from refinement and coarsening requests at different resolutions are integrated with the global solution for the region at every time-step. Thus, using the synchronous AMR approach, a user can explore long-term effects of static policies in a region.

4.4.1 Solution Integration

We devise a simple integration approach to merge solutions at the default resolution of the simulation for the global extent. In regions where coarsening occurs, we interpolate the low-resolution simulation result to the default resolution, and perform map algebra addition to combine it with the global output raster. Similarly, for refinement, we first aggregate the simulation result and perform map algebra addition on the global output raster. Thus, the refinement and coarsening results at different resolutions, in different regions, are integrated in every time-step at the default resolution of the global solution.

Effect of datatype on integration: In case of urbanization outcomes represented by a boolean datatype, we use average, mode or near resampling techniques to merge multi-resolution results at the default resolution (Fig. 9). In case of development pressure represented by a real datatype (e.g., in FUTURES [13]) we adopt one of the two approaches:

(i) recalculate the development pressure over the complete study area after integrating the simulated urbanization results over the global extent or,

(ii) use the result from the highest data resolution simulation in regions with multiple solutions.

4.4.2 Load Balancing

Once again, in the synchronous approach, the master begins by assigning different partitions for simulation at the workers. In every time-step, the workers build and maintain a list of coarsening and refinement requests. Subsequently, these requests are processed at the worker, i.e., a refinement or coarsening request is scheduled for execution at the same worker after the completion of a time-step. Any further refinement required is also carried out at the same worker. Additionally, as part of synchronization, integration of results (see Section 4.4.1) is carried out before the next time-step. Once the solution integration is complete, the worker resumes the simulation on its assigned partition at default resolution for the next time-step. This process is repeated in every time-step for all partitions the study area. Thus, in the synchronous AMR approach, all spatial and temporal interactions are preserved.

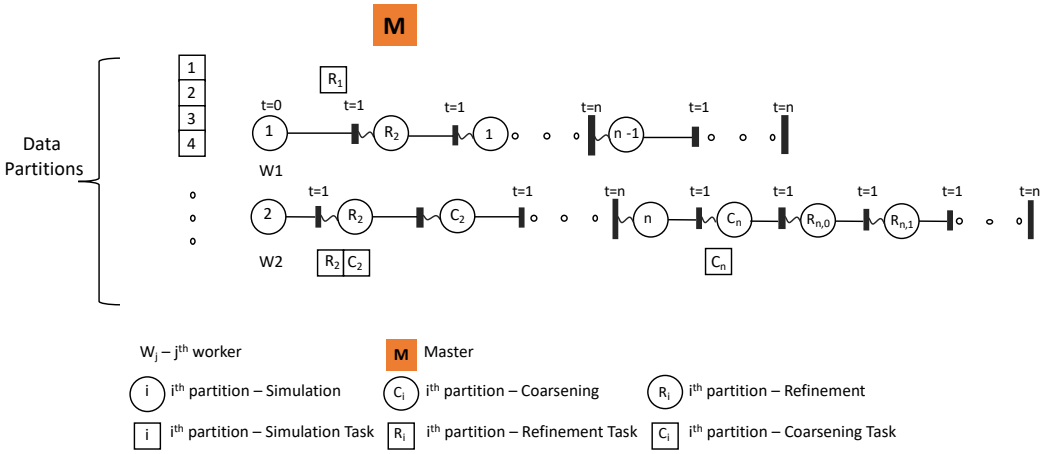


Figure 4 Synchronous AMR – In each time-step, a worker aggregates the refinement and coarsening requests generated during the simulation on its partition. At the end of the time-step, the worker processes these requests and merges their solutions with the global output of the partition at the default resolution.

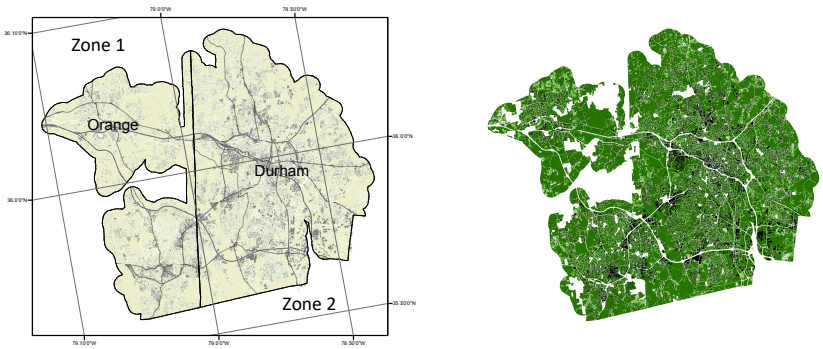


Figure 5 The two zones used in the experiment (left) and the urbanization scene in 2010 (right).

5 Experimental Evaluation

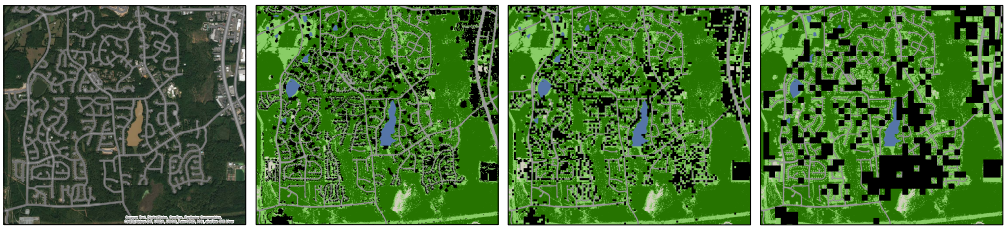
In this section, we describe the experimental setup of our proposed AMR framework. Figure 5 shows the two sub-county zones in the Raleigh-Durham (RDU) region used in our experiments. We carry out our experiments on a system with a hardware spec of 2.5 GHz Intel Core i7 processor and 16 GB memory, and software support for GDAL 2.0 and OpenMPI 1.10. Further, we setup our experiments to use three cores for MPI execution.

Experiment 1: Simulation overhead at different resolutions

In our first experiment, we measure the memory requirement and execution time for a simulation using a fixed input resolution. We setup the study area shown in Fig. 5 to execute 20 time-steps of the simulation in our experiment. We perform three simulation runs, varying the input resolution in each run to use 10m, 30m and 90m input resolution, respectively. Table 1 presents the simulation overhead and Figure 6 illustrates the output maps generated using different input resolution.

Table 1 Simulation Execution Time and Memory Requirement at different input resolutions.

Execution Time (in seconds)			Memory Requirement (MB)		
10m	30m	90m	10m	30m	90m
91.7944	9.2878	1.1584	997	118	12



(a) Satellite Image. **(b)** 10m. **(c)** 30m. **(d)** 90m.

Figure 6 Durham subdivision - Ridges of Parkwood. Fig. 6a is a satellite image from 2017. Fig. 6b, 6c, 6d illustrate urbanization in the year 2030 at 10m, 30m, 90m resolution data, respectively.

Table 2 Simulation Execution Time and Memory Requirement with varying ROI extents.

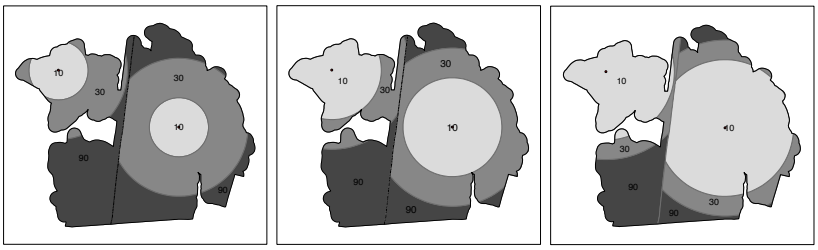
Extent of ROI (in 30m pixels)	Execution Time (in seconds)		Memory Requirement (MB)	
	10m	90m	10m	90m
30 x 30	0.12	0.12	16	15
60 x 60	0.17	0.16	25	21
200 x 200	0.53	0.41	63	39
300 x 300	0.84	0.63	96	48
400 x 400	1.78	1.14	128	55
500 x 500	3.13	1.67	193	61
600 x 600	4.3	2.42	248	64

We observe that both, the execution time and memory requirement increase with use of high-resolution data. Specifically, there is a 9-10x increase in both, the memory requirement and execution time, when the spatial resolution of the simulation is increased by a factor of 3. We use this as a baseline for comparison of the computational improvements in the synchronous and asynchronous approaches in our FUTURES-AMR framework.

Experiment 2: Static Refinement using static policies

In the FUTURES-AMR framework, static refinement supports superimposing finer or coarser meshes in particular regions of interest (ROIs). A static policy for refinement or coarsening defines the exact location and extent of these ROIs for high-resolution or low-resolution simulation. In our second experiment, we measure the overhead to execute refinement (10m) and coarsening (90m) requests with varying ROI extents to test how varying policies would impact computational efficiencies.

We observe that as the size of the ROI increases, execution time and memory requirement for executing a refinement and coarsening request increases. However, the refinement overhead is significantly lesser when compared to using high-resolution 10m data for the simulation over the complete study extent (shown in Table 1). Specifically, in the worst case, a refinement request by a default 30m resolution simulation, increases the the total execution time by 4.3 seconds and peak memory requirement of the simulation by 248MB. Thus, by using the FUTURES-AMR framework for processing refinement and coarsening requests, we incur significantly low computational costs for a multi-resolution simulation.



(a) Policy 1. **(b)** Policy 2. **(c)** Policy 3.

Figure 7 The figure illustrates three policies with varying buffer zones based on two central business districts (Hillsborough in Zone 1 and Durham in Zone 2). In the inner zone, urban development using PGA triggers refinement requests (10m resolution) for patches with patchSize > 15 (patchSize is the total number of 30m pixels to simulate in an urban patch). In the outer zone, urban development using PGA triggers coarsening requests (90m resolution) for patches with patchSize > 30. In the middle zone, urban development using PGA always uses 30m resolution data.

Table 3 Asynchronous AMR - Simulation Execution Time and Number of Requests.

Policy	Resolution		Number of Requests				Time (in s)
	10m	30m	Zone 1		Zone 2		
			Refinement	Coarsening	Refinement	Coarsening	
d2city	< 150	> 350	83	50	6	31	53.57
patchSize	> 15	> 30					
d2city	< 250	> 400	549	13	26	24	133.68
patchSize	> 15	> 30					
d2city	< 350	> 450	683	0	60	16	153.39
patchSize	> 15	> 30					

Experiment 3: Dynamic Refinement using static policies

In our third experiment, we use static policies as illustrated in Figure 7 for dynamic refinement. We run three experiments, where each experiment uses a different policy to simulate urban growth. We begin the simulation using coarse 30m resolution data, switching to high or low-resolution data for patch growth as determined by policy evaluation at runtime. The policies in our experiment specify two attributes for variable resolution simulation:

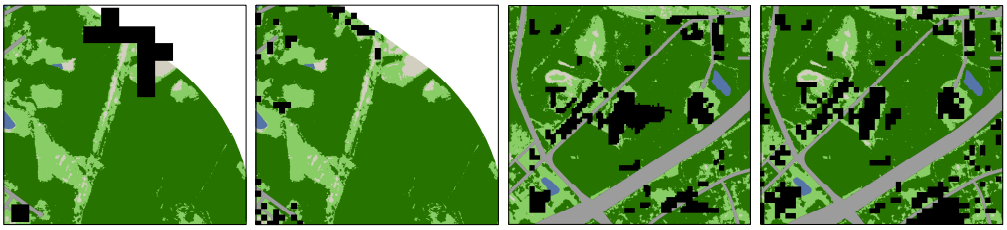
(i) distance of the patch from a central business district (d2city);
(ii) the size of the patch (patchSize).

The attributes are used to define threshold values for coarsening and refinement criteria. In dynamic refinement, the parameter values generated during the simulation are compared against these threshold values to trigger coarsening or refinement. Further, unlike static refinement, additional refinement is triggered if PGA halting criteria is not met. Table 3 and 4 present the measured execution times in the asynchronous and synchronous AMR approaches in our framework with the three policies. Both, d2city and patchSize in Tables 3 and 4 are expressed in terms of number of 30m pixels.

The results indicate that the execution time varies based on the number of requests, which are different between the approaches. Moreover, the execution time for processing different policies vary based on the number of requests. In particular, we observe that total execution time increases with increasing number of requests. Thus, user-defined policies must be carefully selected to limit the adverse impact on the total execution time. Nevertheless,

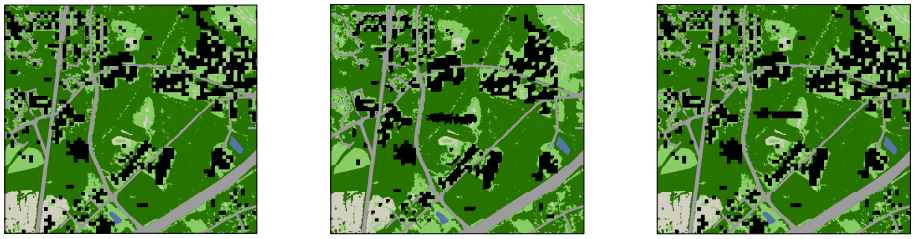
Table 4 Synchronous AMR - Simulation Execution Time and Number of Requests.

Policy	Resolution		Number of Requests				Time (in s)
	10m	30m	Zone 1		Zone 2		
			Refinement	Coarsening	Refinement	Coarsening	
d2city	< 150	> 350	84	51	7	35	85.37
patchSize	> 15	> 30					
d2city	< 250	> 400	639	13	33	26	297.53
patchSize	> 15	> 30					
d2city	< 350	> 450	784	0	63	20	347.35
patchSize	> 15	> 30					



(a) 90m. **(b)** 30m. **(c)** 10m. **(d)** 30m.

Figure 8 Asynchronous AMR - Fig. 8a illustrates an output map of a coarsening request at 90m resolution. Fig. 8b illustrates the output map for the region in Fig. 8a at the default 30m resolution. Fig. 8c illustrates an output map of a refinement request at 10m resolution. Fig. 8d illustrates the output map for the region in Fig. 8c at the default 30m resolution.



(a) 30m. **(b)** 10m. **(c)** 30m.

Figure 9 Synchronous AMR - Fig. 9a illustrates an output map at the default 30m resolution. Fig. 9b illustrates the output map for the region in Fig. 9a for a refinement request at a 10m resolution. Fig. 9c illustrates the composite output map generated by the simulation at the default 30m resolution by combining Fig. 9b and Fig. 9a in the synchronous approach.

the FUTURES-AMR multi-resolution framework demonstrates memory scalability, incurring a maximum additional memory overhead of 248MB as seen in Experiment 2. We also observe that total execution time in the synchronous AMR approach is higher than the asynchronous AMR approach. This increase in execution time is a result of the solution integration approach in the synchronous mode, where results from the multi-resolution simulations at different locations are merged into the final output raster of the study in every time-step. As the asynchronous AMR approach does not merge output results, it performs faster. Finally, in Fig. 8 and Fig. 9, using a few select regions from our study area, we illustrate the effects of user-defined policies on the simulation results generated in the two approaches.

6 Conclusion

FUTURES-AMR has been developed as a computing framework to support multi-resolution geosimulations for use in urban planning and development. In this paper, we described a generic framework for executing a distributed multi-resolution geosimulation and demonstrated its use with the FUTURES geosimulation. We developed static refinement and dynamic refinement techniques with support for expert defined and data-driven polices, along with two new approaches - synchronous and asynchronous AMR for distributed execution of a geosimulation. The results from evaluating the impact of three different user-defined policies on the quality and computational requirements demonstrate the framework's ability to execute a multi-resolution geosimulation with minimal execution time and memory overhead. Thus, in conclusion, the FUTURES-AMR framework, with its support for selective refinement in ROIs is suitable for urban studies using high-resolution data in large study extents.

7 Future Work

Urban development policies are designed in response to urbanization outcomes witnessed in previous years. They have a definitive timeframe associated with them, and often, success or failure of a policy leads to new or modified policies. However, currently, the FUTURES-AMR framework only supports static policies specified a priori. To support dynamic policies in different regions over time, our AMR framework can be integrated with computational steering features that support modification of simulation input at runtime. In future work, we propose to modify our computational steering framework, tFUTURES [22] to allow users to provide dynamic policies as steering input to the simulation.

References

1 Satish Balay, Kris Buschelman, William D Gropp, Dinesh Kaushik, Matt Knepley, L Curfman McInnes, Barry F Smith, and Hong Zhang. PETSc, the portable, extensible toolkit for scientific computation, 1998.

2 J Bell, A Almgren, V Beckner, M Day, M Lijewski, A Nonaka, and W Zhang. BoxLib user's guide. *github. com/BoxLib-Codes/BoxLib*, 2012.

3 Marsha J Berger and Phillip Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of computational Physics*, 82(1):64–84, 1989.

4 Marsha J Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of computational Physics*, 53(3):484–512, 1984.

5 MJ Berger and R LeVeque. Adaptive mesh refinement for two-dimensional hyperbolic systems and the AMRCLAW software. *SIAM J. Numer. Anal*, 35:2298–2316, 1998.

6 P Colella, DT Graves, TJ Ligocki, DF Martin, D Modiano, DB Serafini, and B Van Straalen. Chombo software package for AMR applications-design document, 2000. URL: `http://seesar.lbl.gov/anag/chombo/ChomboDesign-3.1.pdf`.

7 Lori Freitag Diachin, Richard Hornung, Paul Plassmann, and Andy Wissink. Parallel adaptive mesh refinement. In *Parallel processing for scientific computing*, pages 143–162. SIAM, 2006.

8 Anshu Dubey et al. A survey of high level frameworks in block-structured adaptive mesh refinement packages. *Journal of Parallel and Distributed Computing*, 74(12):3217–3227, 2014. `doi:10.1016/j.jpdc.2014.07.001`.

9 Greg L Bryan et al. Enzo: An adaptive mesh refinement code for astrophysics. *The Astrophysical Journal Supplement Series*, 211(2):19, 2014.

10 Joseph E. Flaherty et al. Adaptive local refinement with octree load balancing for the parallel solution of three-dimensional conservation laws. *Journal of Parallel and Distributed Computing*, 47(2):139–152, 1997.

11 Orion S Lawlor et al. ParFUM: a parallel framework for unstructured meshes for scalable dynamic physics applications. *Engineering with Computers*, 22(3-4):215–235, 2006.

12 Peter MacNeice et al. PARAMESH: A parallel adaptive mesh refinement community toolkit. *Computer Physics Communications*, 126(3):330–354, 2000. doi:10.1016/S0010-4655(99)00501-9.

13 Ross K. Meentemeyer et al. FUTURES: Multilevel Simulations of Emerging Urban-Rural Landscape Structure Using a Stochastic Patch-Growing Algorithm. *Annals of the Association of American Geographers*, 103(4):785–807, 2013.

14 Robert D Falgout and Ulrike Meier Yang. hypre: A library of high performance preconditioners. In *International Conference on Computational Science*, pages 632–641. Springer, 2002.

15 Efi Fogel and Monique Teillaud. The computational geometry algorithms library CGAL. *ACM Communications in Computer Algebra*, 47(3/4):85–87, 2014.

16 Daniel A Ibanez, E Seegyoung Seol, Cameron W Smith, and Mark S Shephard. PUMI: Parallel unstructured mesh infrastructure. *ACM Transactions on Mathematical Software (TOMS)*, 42(3):17, 2016.

17 Scott R. Kohn and Scott B. Baden. Parallel software abstractions for structured adaptive mesh methods. *Journal of Parallel and Distributed Computing*, 61(6):713–736, 2001. doi:10.1006/jpdc.2001.1700.

18 John G Michalakes. RSL: A parallel runtime system library for regional atmospheric models with nesting. *IMA Volumes in Mathematics and Its Applications*, 117:59–74, 2000.

19 M Miller. Silo – a mesh and field I/O library and scientific database, 2018. URL: https://wci.llnl.gov/simulation/computer-codes/silo.

20 Manish Parashar and James C Browne. On partitioning dynamic adaptive grid hierarchies. In *System Sciences, 1996., Proceedings of the Twenty-Ninth Hawaii International Conference on,*, volume 1, pages 604–613. IEEE, 1996.

21 Jarmo Rantakokko and Michael Thuné. Parallel structured adaptive mesh refinement. *Parallel computing*, pages 147–173, 2009.

22 Ashwin Shashidharan, Ranga Raju Vatsavai, Abhinav Ashish, and Ross K. Meentemeyer. tFUTURES: Computational steering for geosimulations. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL'17, pages 27:1–27:10, New York, NY, USA, 2017. ACM. doi:10.1145/3139958.3140049.

23 John A Trangenstein. Adaptive mesh refinement for wave propagation in nonlinear solids. *SIAM Journal on Scientific Computing*, 16(4):819–839, 1995.

24 Andrew M. Wissink, Richard D. Hornung, Scott R. Kohn, Steve S. Smith, and Noah Elliott. Large scale parallel structured AMR calculations using the SAMRAI framework. In *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, SC '01, pages 6–6, New York, NY, USA, 2001. ACM. doi:10.1145/582034.582040.