Design and Implementation of “Many Parallel Task” Hybrid Subsurface Model

Khushbu Agarwal, Jared M. Chase, Karen L. Schuchardt, Timothy D. Scheibe, Bruce J. Palmer, Todd O. Elsethagen
Pacific Northwest National Laboratory
902 Battelle Blvd,
Richland WA 99354
{Khushbu.Agarwal, Jared.Chase, Karen.Schuchardt, Timothy.Scheibe, Bruce.Palmer, Todd.Elsethagen}@pnnl.gov

ABSTRACT

Continuum scale models have been used to study subsurface flow, transport, and reactions for many years. Recently, pore scale models, which operate at scales of individual soil grains, have been developed to more accurately model pore scale phenomena, such as precipitation, that may not be well represented at the continuum scale. However, particle-based models become prohibitively expensive for modeling realistic domains. Instead, we are developing a hybrid model that simulates the full domain at continuum scale and applies the pore model only to areas of high reactivity. The hybrid model uses a dimension reduction approach to formulate the mathematical exchange of information across scales. Since the location, size, and number of pore regions in the model varies, an adaptive Pore Generator is being implemented to define pore regions at each iteration. A fourth code will provide data transformation from the pore scale back to the continuum scale. These components are coupled into a single hybrid model using the Swift workflow system. Our hybrid model workflow simulates a kinetic controlled mixing reaction in which multiple pore-scale simulations occur for every continuum scale time step. Each pore-scale simulation is itself parallel, thus exhibiting multi-level parallelism. Our workflow manages these multiple parallel tasks simultaneously, with the number of tasks changing across iterations. It also supports dynamic allocation of job resources and visualization processing at each iteration. We discuss the design, implementation and challenges associated with building a scalable, Many Parallel Task, hybrid model to run efficiently on thousands to tens of thousands of processors.

General Terms

Design, Algorithms, Performance

Keywords

Parallel Tasks, Workflow, Subsurface Modeling.

1. INTRODUCTION

Continuum scale models have been used to study subsurface flow, transport, and reactions for many years. However, they simulate the reaction model in an averaged sense and do not represent subsurface phenomenon at particle level. The pore scale models operate at scales of individual soil grains, and can accurately model phenomenon such as precipitation, and fractures[1] etc., that may not be well represented at the continuum scale. Since a realistic domain may contain on the order of billions of particles, the pore-scale models become prohibitively expensive for modeling a large domain. Developing a hybrid subsurface model that couples the macro-scale model which can simulate the overall system, with the pore-scale model which can accurately represent particle interactions, provides a balance between computation time and model accuracy. We developed the hybrid model, proposed by Scheibe et al [2], which simulates the full domain at continuum scale and applies the pore-scale model only to areas of high reactivity.

Subsurface Transport Over Multiple Phases (STOMP) [3] and Smooth Particle Hydrodynamics (SPH) [4] codes are applied for the macroscale and pore-scale modeling respectively. A separate component is being implemented which identifies the regions of high reactivity which need microscale simulations. Other components are required to provide transformations between the pore and continuum domains. The Swift workflow system [5] is used to develop the model in which tasks execute independently and form a workflow connected using files. An iteration of the continuum scale involves executing multiple independent pore-scale simulations, each of which is a parallel run. The workflow hence follows the *many task computing* [6, 7] paradigm, executing multiple parallel task concurrently.

A tightly coupled model may offer performance benefits but we choose to follow a loosely coupling approach for some important reasons. “First, the two models have very different data structures. Tight integration of shared structure is not required. Second, each of these codes is being developed independently by separate groups and undergoing large-scale development. In addition, the code which determines pore regions, their locations and characteristics would also have to be integrated. Tight integration will require significant effort to manage and be disruptive to ongoing efforts [8].” Finally, adding analysis and visualization methods to the coupled process will further complicate the model.

We present the design and implementation of our “many parallel task” based hybrid model and discuss it in context of our target problem domain. The rest of the paper is organized as follows: section 2 discusses the background and related work. We discuss our target problem in section 3. The different components of the hybrid model are described in section 4 and the design and implementation of workflow are presented in section 5. We discuss our experimental configuration section 6. We conclude and discuss future work in section 7.

2. BACKGROUND AND RELATED WORK

The coupling approach for developing novel models by using multiple independent codes, have been widely used in several domains [9-12]. An MPI [13] based approach to coupling generally requires modifications to existing codes where each component should perform communication steps for the coupled model. The workflow based approaches using scripting languages have been popular due to the ease of implementation maintenance and portability. A coupled model using python as scripting language was developed for multi-physics simulations [9]. The ESSE used shell scripting to develop a workflow for running an ensemble of climate model simulations [12]. The IPS (Integrated Plasma Simulator) framework [10] by Foley et al. used Swift for coupled multi-physics simulation of fusion plasmas.

Several frameworks [14-16] have been implemented over last few years that provide an abstraction from the details of workflow execution, job scheduling, resource management and error handling etc. The Swift workflow language, used to develop the hybrid subsurface model, offers an implicitly parallel and deterministic programming model [17], which is central to our multi-parallel task based workflow design. It also provides functional mappers, which allows external applications to be applied to file collections. Moreover, a C-like syntax and abstraction from complex details of parallel execution greatly simplifies the implementation process.

2.1 Swift

A Swift script describes data, application components, and invocations of applications. A Swift workflow generally involves executing a large number of independent tasks in an HPC or distributed environment. The advantages to using Swift include an elegance of remote execution that is inherent to the language. Swift also provides file and data management capabilities. The *file mapper* constructs are used to specify disk-resident data. The Swift mappers use an expression language to query for output files produced by the simulation then groups each file set into an array structure so they can be managed together.

Swift has an inherent parallel nature; when iterating over arrays, each task is performed in parallel. This makes the execution of parameter studies much more efficient. Also, the complexities of parallelization are encapsulated. This makes the launching of multiple remote jobs in parallel and the monitoring their status, simple to implement. Swift will launch every job in parallel and wait for them to finish. Once each job is complete Swift will check for the expected output files.

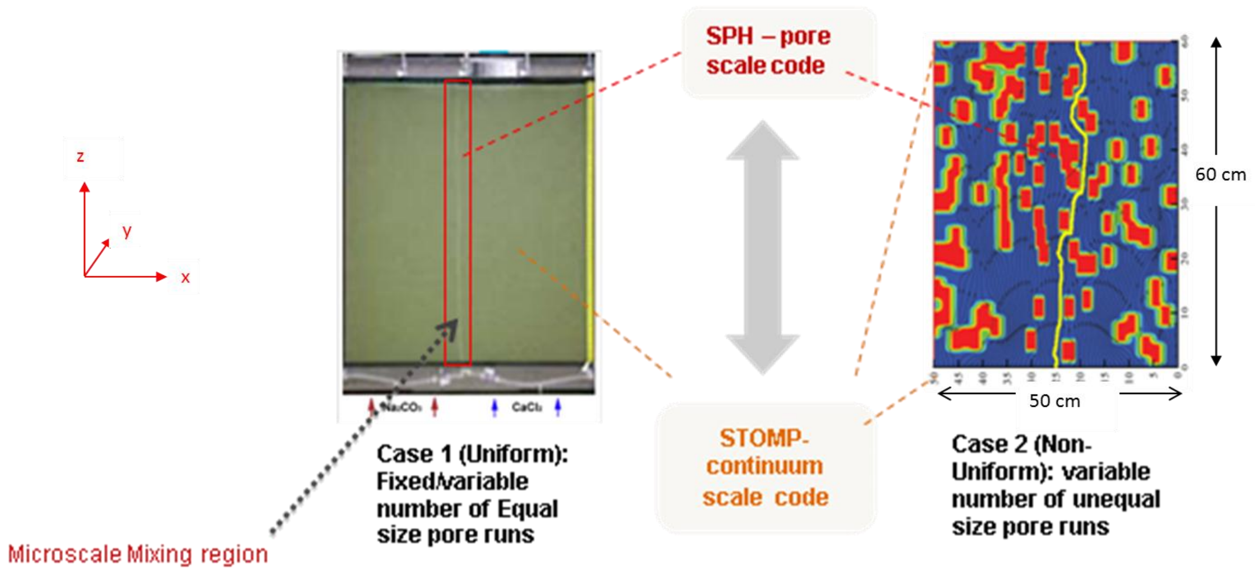

z
y
x
SPH – pore scale code
60 cm
50 cm
Na_2CO_3
$CaCl_2$
Case 1 (Uniform): Fixed/variable number of Equal size pore runs
STOMP- continuum scale code
Case 2 (Non-Uniform): variable number of unequal size pore runs
Microscale Mixing region

Figure 1: Modeling Target Problem Domains using STOMP and SPH

MTAGS'11, November 14th, 2011, Seattle, WA, USA.
Copyright © 2011 ACM 1-58113-000-11/14/2011…$10.00.

3. PROBLEM DESCRIPTION

Our hybrid subsurface model developed by Scheibe et al [2], will simulate the parallel transport of two solutes with a mixing-controlled kinetic precipitation reaction occurring at the interface between the two solutes. The system is filled with porous medium like sand. The sand is saturated with water, flowing from bottom to top at a specified volumetric flux rate (corresponding to a specified average pore velocity). As the solutions flow upward through the flow cell, they mix along the centerline by diffusion, leading to super-saturation and precipitation of calcium carbonate mineral. The precipitated mineral phase modifies the pore geometry, blocking pores and inhibiting mixing of the two solutes. Therefore there is a strong coupling/feedback at the pore scale that strongly impacts macroscale behavior. The coupling approach will utilize the Dimension Reduction with Numerical Closure (DRNC) method as presented by Tartakovsky [18].

Use Case 1 (Uniform - Equal size pore scale runs): The system is filled with uniform sand. Thus, the mixing region is known and can be divided into equal size pore domains [Figure 1: Use Case 1)]. At the beginning of the simulation, the solutes are injected from bottom. Initially, there will be no mixing at the top of the pore domain so pore models will not be run in this location. Likewise, as the simulation proceeds, the mixing will cause precipitate to form at the bottom and no further reaction may occur so these areas may not require detailed simulation later. The simulation zone will be like a moving window that moves up the system.

Use Case 2 (Non-Uniform Variable size pore scale runs): The experimental system is packed heterogeneously with two different types of sand having different hydraulic conductivities and porosities (red is low conductivity and blue is high)[Figure 1: Use Case 2], thus the path of two solutes is indeterminate. The mixing zone will vary, resulting in non-uniform unequal sized pore domains.

We target the uniform use case as our initial problem domain. The pore-scale region in this case can be modeled as a single pore domain. However this would not be extensible for the generic case. Hence, we implement it as MTC model.

4. HYBRID MODEL WORKFLOW

The workflow consists of four main modules [Figure 2]. We are using a serial version of the macroscale code (STOMP) to model the full problem domain though for a larger domain, a parallel version can be inserted into the workflow seamlessly. The code that determines pore regions (PG) and the algorithm to calculate mixing coefficient (GPG) are serial. The pore-scale simulations (SPH) will involve millions of particles and requires parallel execution for each region. The pore domain will be fixed for the uniform case when modeling it as a single SPH run; and variable otherwise. Each of the SPH domain will be equal in number of particles for the uniform case and unequal otherwise [Section 3].

A complete simulation will involve executing many iterations of the hybrid model. The workflow components are summarized in the following sections, with an emphasis on changes required to support the hybrid model in the case of the existing codes.

4.1 STOMP

The STOMP [3] code models the macro-scale reaction. Since the concentrations of particles in the pore-scale domain changes after the last macro-scale calculations have been performed, the STOMP code was modified to incorporate mixing coefficient calculations. If a mixing coefficient file is present, the stomp calculates initial concentrations based on the mixing coefficient of the particular grid and the STOMP restart file. A default value is used otherwise, in which case the initial concentration are assumed to be same as that in the restart file produced at the end of previous STOMP simulation.

4.2 Pore Generator

The Pore Generator (PG) is a python program that provides

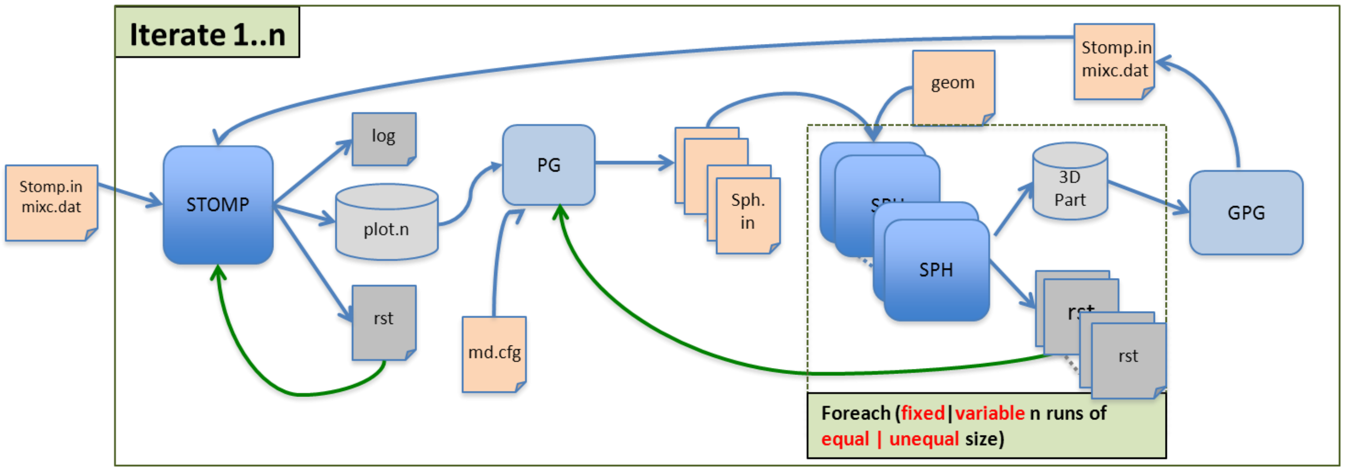


Figure 2: Hybrid model workflow, depicting the components and data flow

mathematical model to calculate particle properties from macroscale to the microscale (pore) domain. The PG consists of two modules : 1) An Adaptivity Manager determines how many and which pore-scale subdomains require simulation and 2) A Pore Concentration Generator which performs the “reconstruction” of the pore-scale initial conditions for pore-scale subdomains.

4.2.1 *Adaptivity Manager:* A pore-scale subdomain is considered active when the average of initial concentrations of the fluid particles reaches a user specified minimum threshold value. The average concentration of the pore-domain is obtained from the STOMP output. Let for any pore-scale domain *‘i’:*

[A] = average concentration of solute A

[B] = average concentration of solute B, and

Ksp = solubility constant

We define a pore-scale domain to be active when,

$$[A]*[B]/K_{sp} > 1.01,$$

For the first time step, the number of pore-scale domains is $N_{ps} = 1$. The N_{ps} parameter is incremented each time one or more pore domains are added.

4.2.2 *Pore Concentration Generator:* This module performs the “reconstruction” of the pore-scale initial conditions for N_{ps} pore-scale subdomains. The initial concentration of particles in the pore-scale domain are calculated based on output from STOMP, and pore-scale output from the previous time step. The boundary conditions are dependent not only on the previous SPH runs for that domain but also on neighboring pore-scale domains. For e.g., for use case 1:

- For iteration 1, particles near the inlet boundary in the bottom(z=0) SPH domain, are assigned concentration A=1 on left half, and B =1 on right half.
- The particles near the inlet boundary are assigned concentrations from the adjacent uppermost grid cell of the neighboring SPH domain.

$$A_i^{n+1} = A^n_{klm} \text{ for all fluid particles i, for } z_i < z_{BC}$$

- The remaining particle concentrations are assigned according to:

$$A_i^{n+1} = c_A A^n \text{ for all fluid particles i, for } z_i > z_{BC,}$$

where,

$$c_A = \frac{A_{avg}N_{pf} - \sum_{z<z_{BC}} A_i^{n+1}}{\sum_{z>z_{BC}} A^n}$$

n = iteration number, A= particle concentration, i = id of fluid particle, N_{pf} = number of fluid particles, A_{avg} = Average concentration of grid cell from macroscale, z= vertical distance of particle from pore domain boundary, z_{bc} = user defined percentage threshold for boundary condition.

4.3 SPH

At the microscale, SPH code is used to model particle interactions. The current SPH code [4] requires some modifications in order to completely integrate it into the multiscale coupling scheme. The “A+B ->C” reaction model needs to be modified to support a new boundary condition corresponding to injection of A and B from the bottom of the system. In addition, particles that flow out of the top of the system and are re-injected at the bottom must be assigned proper values. A nearest neighbor approach will be used to determine these values. To support future complete modeling of the precipitation reaction, a new chemistry module that supports direct modeling of the precipitate will be developed.

4.4 Grid Parameter Generator

The Grid Parameter Generator (GPG) is a python script that creates STOMP input files based on output from SPH runs. During the invocation of GPG, a list of SPH outputs mapped to their corresponding STOMP grid cell are passed as input.

The GPG parses output files from the last time step of every SPH job to obtain the concentrations of each constituent, A and B, at each fluid particle in the reaction chamber. These concentrations are used to calculate a mixing coefficient using the following equation.

$$m = \frac{\overline{AB}}{\overline{A}\overline{B}} = N_p \frac{\sum_{i=1}^{N_p} A_i B_i}{\sum_{i=1}^{N_p} A_i \sum_{i=1}^{N_p} B_i}$$

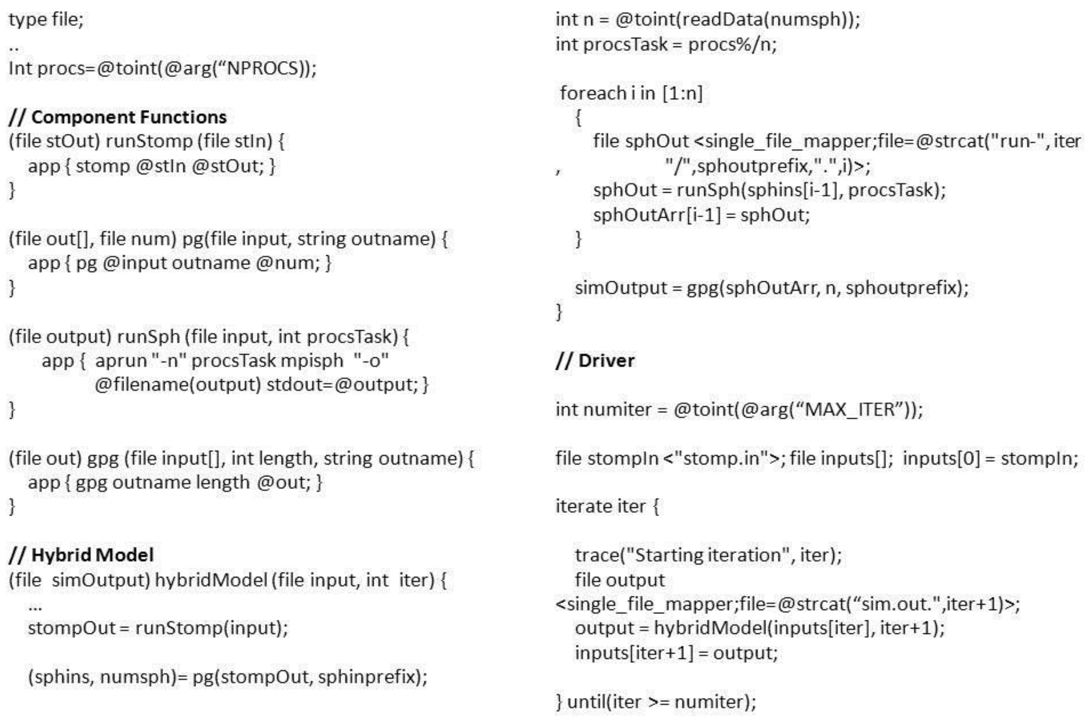
Note that N_p is the number of fluid particles (not including solid particles).

A file of values of *m* for each STOMP grid cell is then generated in the STOMP file format, in which *m* for those STOMP cells corresponding to an SPH simulation domain are obtained from the computation above applied to the SPH output, and m for all the other STOMP cells set to a value of 1.0. The STOMP code has been modified to read the mixing coefficients and modify the internal reaction rates.

5. DESIGN AND IMPLEMENTATION

Our hybrid subsurface model, implemented as a multi-parallel task workflow is presented in Figure 3. The workflow is adaptive and portable. It supports dynamic scheduling of tasks, and utilizes Swift’s logging and error handling capabilities. We add visualization and provenance capabilities in the workflow which will allow scientists to track the overall simulation during and after the run. The hybrid model workflow is launched by invoking a single instance of Swift, along with request to allocate all the resources needed during the execution. A typical job launch script would contain request for resources that is typical to the underlying system, along with command to launch Swift as a serial controller that manages the compute tasks. As the resources are acquired at the beginning of the execution, it eliminates the need to go through job queue multiple times. The number of iterations to be executed and the number of processes available are specified at the command line.

The Swift workflow requires simple methods to act as an interface to system calls. Such functions are implemented as '*app*'. Due to temporary issues in Swift's filesys_mapper, we design the PG module into two separate components. The adaptivity manager first specifies number of pore regions (N_{ps}), the workflow then uses it to construct an array of length N_{ps} and alleviating the need to use arbitrary array size. We create an '*app*' [Figure 3] each for the stomp(), pg_adaptivity_mgr(), pg_conc_gen(), sph() and gpg() corresponding to the different components of the model as discussed in Section 4. The app interface is required to clearly identify the input and output parameters and files. Since, swift is designed to execute in separate independent workspace than the current work directory, the files that are not specified as I/O for an app are not moved back and forth between the two and may get lost. The Hybrid model function issues first call to STOMP, which generates an array of output files. The file written at the last STOMP time step is used as an input to the PG's adaptivity manager. This produces a output file which specifies number of pore-scale runs 'N' needed for the particular iteration and the location of the pore-regions. The subsequent call is made to PG's concentration generator which prepares the input file, containing the concentration of each pore-scale grain, for each of the N pore-scale simulations.


```
type file;
..
Int procs=@toint(@arg("NPROCS));

// Component Functions
(file stOut) runStomp (file stIn) {
   app { stomp @stIn @stOut; }
}

(file out[], file num) pg(file input, string outname) {
   app { pg @input outname @num; }
}

(file output) runSph (file input, int procsTask) {
     app {  aprun "-n" procsTask mpisph  "-o"
              @filename(output) stdout=@output; }
}

(file out) gpg (file input[], int length, string outname) {
   app { gpg outname length @out; }
}

// Hybrid Model
(file  simOutput) hybridModel (file input, int  iter) {
   ...
   stompOut = runStomp(input);

   (sphins, numsph)= pg(stompOut, sphinprefix);

int n = @toint(readData(numsph));
int procsTask = procs%/n;

 foreach i in [1:n]
   {
     file sphOut <single_file_mapper;file=@strcat("run-", iter
,               "/",sphoutprefix,".",i)>;
     sphOut = runSph(sphins[i-1], procsTask);
     sphOutArr[i-1] = sphOut;
   }

   simOutput = gpg(sphOutArr, n, sphoutprefix);
}

// Driver

int numiter = @toint(@arg("MAX_ITER"));

file stompIn <"stomp.in">; file inputs[];  inputs[0] = stompIn;

iterate iter {

   trace("Starting iteration", iter);
   file output
<single_file_mapper;file=@strcat("sim.out.",iter+1)>;
   output = hybridModel(inputs[iter], iter+1);
   inputs[iter+1] = output;

} until(iter >= numiter);
```

Figure 3: Swift Workflow for the Hybrid Subsurface Model

The pore-scale simulations are launched using the Swift's "foreach" construct which executes the tasks in parallel. Each of these SPH runs is launched as an MPI [11] job. Our current implementation assumes that the size of the pore-regions are nearly equal and can be modeled in similar time. Hence, the processes are divided equally between the pore simulations. Once all pore-scale runs complete, the GPG uses the output files to generate mixing coefficients which feeds back into STOMP. The main driver of the workflow executes each iteration serially using Swift's "iterate" construct.

5.1 Adaptive and Dynamic Scheduling

The number of pore-scale simulation varies between iterations as the reaction travels upward through the system. Once the number of pore-scale regions has been determined for a given iteration, the pore-scale simulations are launched by evenly distributing available processes among the pore-regions. For the non-uniform case, this may lead to load balancing issues. A weighed scheduling policy can be implemented to address the load imbalance, however this may require identifying number of particles or other features of each pore-domain prior to launch.

In case the number of pore-regions is more than the available processes, the workflow queues the pore-tasks and launches them in multiple batches. This allows for efficient and maximum utilization of the available resources. In the worst case scenario (number of processes =1), all pore-scale simulations can be launched as serial job in an iterative manner.

5.2 Portability

Our Swift workflow is portable, the system specific details being provided through the Swift configuration file. A configuration script identifies the underlying resource manager and prepares the task launching command. For example, on a system running PBS, the parallel pore tasks are launched using an 'aprun' command while for a machine using SLURM resource manager, the tasks are launched using srun or mpirun. Swift itself is configured to run on the login node thus bypassing any issues with invoking tasks from compute nodes, which is not supported on all systems.

Note that our Swift workflow does not use the recently developed JETS [5] functionality in which the tasks are managed by MPICH based task manager. The workflow simply uses Swift as a scripting language and manages the parallel tasks explicitly in the workflow. We plan to evaluate our workflow with JETS task manager as it becomes available with support for multiple MPI tasks.

5.3 Data Management

The hybrid model workflow produces tremendous amount of data. Swift's 'file mappers' and arrays provide a simple interface to perform pattern matching against named files on disk. By specifying an expression on output files for the run, the desired files are placed into an array automatically. If there are no files produced that match the expression specified in the script, then an exception is thrown and the user is notified. The error handling capabilities described occur as part of the language and do not have to be written within the script.

Swift also simplifies data management by implicitly removing files that are not specified as part of the workflow. However it necessitates that all files that might be needed for provenance/visualization or other data analysis capabilities are identified in the workflow in order to be preserved. Swift also ensures that each swift job executes in an independent work directory, hence multiple jobs can be submitted without risk being over-written. The hybrid model itself is designed to map every pore-scale run to a different directory, which are identified by a unique pore-id.

5.4 Visualization

Visualizations of SPH output are performed using Visit. Visit is invoked through the Swift workflow and passed a

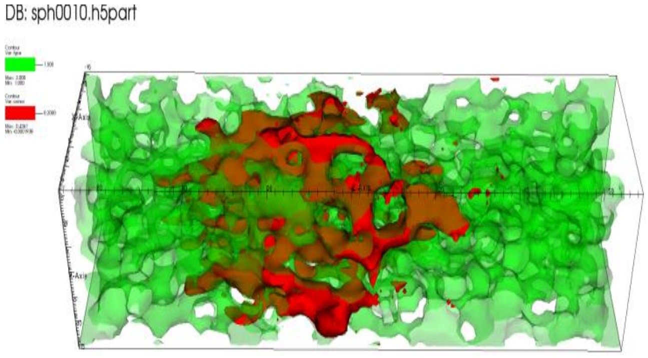


Figure 4: SPH visualization

python script that contains instructions for automating the generation of 3D visualizations from SPH output. Initially, visualizations will be performed at every iteration, it can be modified to control the frequency of image generation. The SPH visualization [Figure 4] shows a contour of liquid particles (green) with concentrations of the constituent (red) flowing through the medium. Note that the brown color represented is actually the constituent (red) flowing behind the liquid particles (green).

Visualizations of STOMP output [Figure 5] are also created through the swift workflow using Visit. After a STOMP simulation has completed, Swift invokes a data translation script to convert the output to tecplot format. This is then used as an input to the Visit tool to generate the plot. Note that visit will support both 2D and 3D data produced by STOMP.

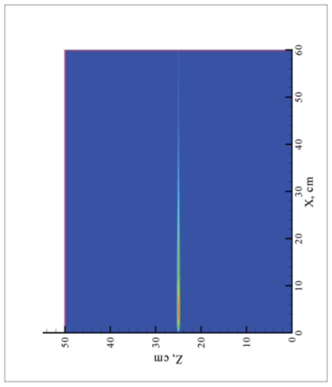


Figure 5: STOMP Visualization for Target Problem

Figure 5 shows the STOMP visualization for the test case when Na2CO3 and CaCl2 are injected from two different sides at the bottom. The thin multi-colored line where z = 25 shows where the two constituents are reacting and forming a precipitate.

5.5 Provenance

Provenance information is important for monitoring simulations in real time, post-analysis, and for long term tracking and knowledge capture of the modeling process. In the case of the hybrid model, additional Swift apps dedicated to the capture of provenance information are added to the workflow. These apps are responsible for synthesizing and recording provenance in a format suitable for ingest into the SALSSA user environment [21] which will be used to run the hybrid model. Currently SALSSA expects provenance to be represented as a Resource Description Framework (RDF) serialization of the Open Provenance Model (OPM) [22] for ingest into its provenance repository. Provenance collected by the apps includes: iteration number, start/end time for iteration, number of pore regions per iteration, location of pore regions per iteration, size of pore regions per iteration, result of last time step per simulated pore region, and geometry file(s) used by each iteration.

These provenance apps are designed and customized specifically for supporting provenance requirements for the hybrid model use case. However, it should be possible to provide provenance capture techniques for use within the workflow scripting environment itself, supporting a more general approach. For example support for different provenance recording serializations, support for metadata extraction rules to harvest provenance information from selected input and output files, support for identification of file types to include or exclude in the provenance capture, and selection of processes (i.e. apps) to include or exclude in the provenance capture.

For the hybrid model, provenance will only be collected at the STOMP iteration level. Normally SALSSA would expect provenance relationships to be captured for each SPH process as well. However, due to the large amount of provenance this would generate, its impact to workflow performance, and arguably the lack of usefulness this would have for a user it was decided to summarize SPH provenance information for each iteration. This would include geometry file(s) used and information regarding the simulated pore regions (e.g. size and location). Given this information, SALSSA could provide a real time visualization of the simulation's progress, as it moves from one iteration to the next.

6. EXPERIMENTAL CONFIGURATION

Our initial target problem domain consists of a 50 cm (wide)* 60 cm (height) 2D system. The system is filled with uniform sand, Na_2CO_3 and $CaCl_2$ are injected from two different sides at the bottom. As the mixture is homogeneous in porosity, the two solutes are expected to diffuse at the centerline (x = 25). The STOMP grid is uniform, 0.5 mm in each direction. It's designed to ensure a single grid line is centered at the middle of the domain and wide enough so that a single center grid line can then be used to model the pore-scale phenomenon. The number of pore-scale regions (grid cells) (0.5 * 0.5 mm) can vary from

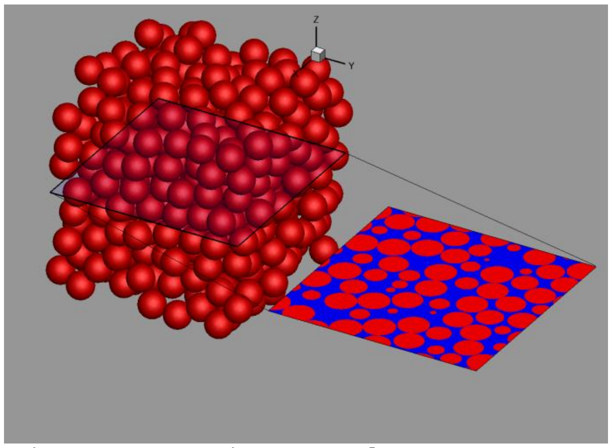


Figure 6 - Generating pore-scale geometry: A 2D geometry is generated by slicing a plane through 3D.

1 to 120.

We target NERSC's Franklin system to demonstrate the hybrid model workflow. We implement and test the Swift workflow system using stubs for different components as the test case geometries and other model specific input files are prepared. The PG module is currently under development too. A 3D periodic pore-scale geometry is generated to ensure all pore-scale simulations were consistent with the complete domain configuration, and a 2D slice was extracted [Figure 6] for demonstration of the model.

For our target problem domain (homogeneous use case), the STOMP (macroscale) simulations are expected to complete in order of seconds. Each of the pore-scale regions will contain nearly 5 million particles. The workflow will execute on the order of 1-120 of these SPH runs in parallel for each iteration, and a total of ~100 iterations of the model. The SPH code has been demonstrated to take 2 minutes 36 seconds for a 3.4M particle simulation with 1K processes. On average we expect each of the SPH run to be executed using 128 processors, and to take between 10-15 minutes for execution. The serial components are not expected to take significant time. We incorporate parallel I/O for SPH to reduce the read/write overhead but additional methods may be needed. Thus, the model is expected to take around 30-40 hours for the complete run. The computing costs will increase significantly for a complete 3D simulation and less uniform problems.

We plan to run the model on nearly 10K processes for the initial 2D simulation. As the number of pore-regions is expected to increase with iterations, the number of processes for each SPH run will decrease as we move further along in the simulation. We need to introduce changes to the workflow to ensure each SPH simulation should run for a certain number of processes only. For e.g, during the first iteration, the workflow may not be able to successfully execute single SPH job on 10K processes.

7. CONCLUSION AND FUTURE WORK

In this paper we have presented the design of our portable, MTC based hybrid subsurface model. The model is implemented using a loose coupling approach, with Swift workflow language. The workflow has been tested with stubs using similar interface as by the different components of the hybrid model. The model targets the use case where all parallel tasks can be assumed to take similar amount of time to complete, however its generic and can be used in non-uniform problem domains as well. The load-imbalance problem may be addressed by incorporating weighed scheduling for less-uniform problem domains.

The model uses Swift"s data management, error handling and inherent parallel model execution capabilities. The workflow is expected to undergo minor modifications as complex problem domains are used, requiring additional data mappings. Reducing I/O overhead may require incorporating novel solutions like RAM disk architectures. Additional data analysis techniques for the model may be added as well, per scientific requirements. Minor modifications will be made to the workflow, to threshold the visualization and provenance data generated, and enable them only at selected iterations. We also plan to integrate the Swift workflow with the SALSSA framework, which will provide an abstract interface for launching the hybrid subsurface model on different systems.

8. ACKNOWLEDGMENT

We would like to thank Michael Wilde and Justin Wozniak at Argonne National Laboratory for their help and support with Swift workflow system.

This work was under Office of Science, SciDAC(Scientific Discovery through Advanced Computing) project.

9. REFERENCES

[1] Tartakovsky, A. M., P. Meakin, T. Scheibe, and B. D. Wood, "A smoothed particle hydrodynamics model for reactive transport and mineral precipitation in porous and fractured porous media," Water Resources Research, 43(5):Art. No. W05437.

[2] Scheibe, T. D., A. M. Tartakovsky, D. M. Tartakovsky, G. D. Redden, and P. Meakin, 2007. Hybrid numerical methods for multiscale simulations of subsurface biogeochemical processes, Journal of Physics: Conference Series 78: 012063.

[3] Nichols, W.E., N.J. Aimo, M. Oostrom, and M.D. White. 1997. STOMP Subsurface Transport Over Multiple Phases: Application Guide PNNL-11216 (UC-2010), Pacific Northwest National Laboratory. http://stomp.pnnl.gov/documentation/guides/application.pdf

[4] Palmer, B. J, Y. Fang, G. E. Hammond, and V. Gurumoorthi, 2007. Component-based framework for subsurface simulations, Journal of Physics: Conference Series 78:012047, doi:10.1088/1742-6596/78/1/012047.

[5] Wilde, M., Foster, I., Iskra, K., Bechman, P., Zhang Z., Espinosa A., Hategan M., Clifford B., Raicu I. 2009. Parallel Scripting for Applications at the Petascale and Beyond, Computer, Vol. 42, No. 11.

[6] Ioan Raicu, Ian Foster, Yong Zhao. “Many-Task Computing for Grids and Supercomputers”, IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08). Vol. no. pp.1-11, 17-17 Nov 2008

[7] E. Ogasawara, D. de Oliveira, F. Chirigati, C. E. Barbosa, R. Elias, V. Braganholo, A. Coutinho, and M. Mattoso, “Exploring Many Task Computing in Scientific Workflows,” in MTAGS ’09: Proc. of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, Nov 2009.

[8] Schuchardt, K. L., Palmer B., Agarwal K., Scheibe T. 2011. Many Parallel Task Computing for a Hybrid Subsurface Model. SciDAC 2011.

[9] J. U. Schlüter, X. Wu, S. Kim, S. Shankaran, J. J. Alonso, and H. Pitsch, A Framework for Coupling Reynolds-Averaged with Large Eddy Simulations for Gas Turbine Applications, *Journal of Fluids Engineering,* 127(4):608-615, 2005.

[10] Foley, S.S.; Elwasif, W.R.; Bernholdt, D.E.; Shet, A.G.; Bramley, R.; 2010. Many Task Applications in the Integrated Plasma Simulator, IEEE Workshop on Many Task Computing on Grids and Supercomputers (MTAGS) 2010.

[11] P. Marshall, M. Woitaszek, H. M. Tufo, R. Knight, D. McDonald, and J. Goodrich, “Ensemble Dispatching on an IBM Blue Gene/L for a Bioinformatics Knowledge Environment,” in MTAGS ’09: Proc. of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, Nov 2009.

[12] C. Evangelinos, P. F. Lermusiaux, J. Xu, P. J. Haley, C. N. Hill, “Many Task Computing for Multidisciplinary Ocean Sciences: Real-Time Uncertainty Prediction and Data Assimilation,” MTAGS ’09: Proc. of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, Nov 2009.

[13] Message Passing Interface Forum, “MPI: A message passing interface standard version 2.2.” http://mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf, September 2009.

[14] R. Costa, F. Brasileiro, G. L. Filho, and D. M. Sousa, “OddCI: Ondemand distributed computing infrastructure,” in MTAGS ’09: Proc. Of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers,2009.

[15] I. Raicu, I. Foster, M. Wilde, Z. Zhang, K. Iskra, P. Beckman, Y. Zhao, A. Szalay, A. Choudhary, P. Little, C. Moretti, A. Chaudhary, D. Thain, “Middleware support for many-task computing,” Cluster Computing, vol. 13, no. 3, pp. 291–314, 2010.

[16] L. Hui, Y. Huashan, L. Xiaoming, “A Lightweight Execution Framework for Massive Independent Tasks,” in IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08), Nov 2008.

[17] Wilde, M., Hategan M., Wozniak J. M., Z., Clifford B., Katz D. S., Foster I., Swift: A language for distributed parallel scripting. ◦

[18] Tartakovsky A. M., A Panchenko, and KF Ferris. 2011. Dimension reduction methods for multiphase flow and reactive transport in porous media."

[19] Wozniak, J. M., Wilde M., JETS: Language and System Support for Many Parallel Task Computing. Preprint ANL/MCS-P1885-0411, April 2011.

[20] MPICH2:http://www.mcs.anl.gov/research/projects/mpich2/index.php

[21] Schuchardt, K., J. Chase, J. Daily, T. Elsethagen, B. Palmer and T. Scheibe, 2009. Application of the SALSSA framework to the validation of smoothed particle hydrodynamics simulations of low Reynolds number flows, Journal of Physics: Conference Series 180: 012065.

[22] Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E. and Van den Bussche, J. (2010) The Open Provenance Model core specification (v1.1). Future Generation Computer Systems.