1 Editor Comment #1

Dear authors,
In my role as Executive editor of GMD, I would like to bring to your attention our Editorial version 1.1:
http://www.geosci-model-dev.net/8/3487/2015/gmd-8-3487-2015.html
This highlights some requirements of papers published in GMD, which is also available on the GMD website in the Manuscript Types section:
http://www.geoscientific-model-development.net/submission/manuscript_types.html
In particular, please note that for your paper, the following requirement has not been met in the Discussions paper:

- "The main paper must give the model name and version number (or other unique identifier) in the title."
- If the model development relates to a single model then the model name and the version number must be included in the title of the paper. If the main intention of an article is to make a general (i.e. model independent) statement about the usefulness of a new development, but the usefulness is shown with the help of one specific model, the model name and version number must be stated in the title. The title could have a form such as, Title outlining amazing generic advance: a case study with Model XXX (version Y).

In order to simplify reference to your developments, please add a version number and consider to add the models name acronym in the title of your article in your revised submission to GMD.
Yours,

Astrid Kerkweg

We thank the executive editor for catching this issue, and have accordingly added the ISSM version number to the title along with the acronym for ISSM.

2 Reviewer #1:

General comments:

This paper works to integrate the Ice Sheet System Model (ISSM) into an interactive online model that can be accessed by a larger community. It transforms a series of existing MATLAB and Python classes, currently used by ISSM users to generate model runs, into equivalent JavaScript classes, enabling direct deployment in a web environment. It leverages commercial cloud virtual machine and web service technologies to enable rapid generation of ISSM results via user adjustments of model parameters within a web browser.

This work represents an important step forward in bringing the powerful capabilities of Earth System Models to a larger community. Conventionally, a researcher interested in exploring such a model must invest considerable time to learn FORTRAN or C++ codes, or if they are lucky, there are MATLAB or Python scripts that are provided as wrappers around the lower level codes. Nevertheless, users must still invest much time to handle protocols surrounding input data formatting, methods for parameter adjust- ment, and other methods for controlling the model. Deploying such a model in a web environment will enable a wide new range of model exploration to take place.

We thank the reviewer for the positive assessment of the manuscript, and for the time spent giving advice on how to improve the flow of the text. We have tried to improve the presentation of the new methods, to better frame for which audience they are relevant, and for what type of computational scenarios the new JavaScript API is used. This was a concern echoed also by reviewer 2. Modifications were therefore made in particular to the introduction.

Specific comments:

Para beginning line 132: this paragraph is unclear. Is savemodel unique because it occurs only in the MATLAB implementation? Is this a mechanism for visualizing local simulations set up using MATLAB? Please clarify.

We have reformulated this paragraph to explain better the philosophy behind the "savemodel" routine, the fact it's unique to MATLAB, and allows to shorten the turn-around between running a MATLAB based scientic simulation in ISSM, and transferring its results, along with the underlying "model" class, to a webpage using a JavaScript include file.

Lines 139-150: Until this section, I understood that existing MATLAB and Python pre- configuration wrappers were being converted to JavaScript, but not the main ISSM code. However, this section discusses converting everything to JavaScript. Later, it becomes clearer that the core ISSM code is used in a parallel configuration on EC2 for the larger, continental scale model runs (presumably the C++ version?). I suggest setting this up more clearly earlier in the manuscript, explaining the two primary im- plementations and the reasons for each approach. Presumably the EC2 simulations would involve a user setting a simulation to run and the results being returned after some time? How would that be handled in the web environment (e.g. the user is emailed when results are ready?).
We understand the confusion, and have strived to clarify this point (see reviewer 2's remark on the subject too) in the paragraph by reworking this aspect extensively. We now clearly differentiate the C++ core (used for large-scale model runs on the Amazon EC2 instances) and the JavaScript core (equivalent of the C++ core but translated into JS using emscripten), used for computations local to the webpage that do not require parallel computing (small models). As

suggested, we also introduce this point earlier on in the introduction.

Line 217: on the assumption that the audience may not know a lot of glaciology, I suggest explaining SMB, transient ice flow, etc in more general terms (e.g. surface climate forcings, etc).

We took the reviewer's advice and simplified the text to make it more accessibility to a larger audience.

Line 222: this is providing a bit more clarity on the savemodel component, but I am still not discovering the breakthrough it enables. Is there some kind of caching of initial output being done here to speed up the implementation in the web browser?

We have tried to clarify this statement. The breakthrough is really in shortening the turn around between running a science study with ISSM, and getting it transferred to a JavaScript environment. It's not related at all with caching results, as also suggested by reviewer 2, but just deals litterally with sending the model object to a new environment, a WebPage. We tried and modify the manuscript accordingly.

Fig 3 caption: fix: . . .shows the Columbia Glacier webpage is, ... Again, quite a bit of glaciology jargon here.

Fixed the typo and simplified/improved the text further.

Fig 2: It is good to see a conceptual map like this, but some additional detail could help. The labeling has considerable amount of acronyms. A few simple terms identifying that the client is on the left, the server on the right, and the flow of input/output through the diagram, would help, especially for those not immediately aware of all the different terms.

We have considerably improved the figure by taking the reviewer's advice, and recasting the caption to introduce the relevant acronyms instead of having them on the figure itself. We generalized the terminology to make it more accessible to a general audience.

3 Reviewer #2

General comments:

The paper presents the development of a Javascript Interface for the Ice Sheet System Model, aimed at simplifying the interaction and running of the model for less specialist users. Making modelling and the results of modelling more

accessible is key to aiding wider understanding of the research. The paper is largely well written, therefore, I recommend it for publication, however, there are a few things that could be clarified.

The main thing that isnt clear to me is who the API is targeted at on page 4 it reads as if the ISSM experts will still be setting up the model runs, and the API is mainly a tool for communicating the results of the modelling to the wider community and possibly the public. In the examples you have in the VESL, these are relatively simple (I realise they are demonstrators) and all the scenarios (e.g. SMB change) could be pre-run and the API is then just a tool to allow the user to engage with the results, it doesnt need the model to be re-run by every user, this seems to be a waste of computational resource. Even in more complex scenarios, it still seems like pre-running the simulations and making just the results available for interrogation would be more efficient. But - is this what is happening is this what is meant on line 223? Are the results on the VESL all pre-run? This really isnt clear to me.

We thank the reviewer for the effort in the review, the acknowledgement of the importance of making modeling and the results of modeling more accessible to a wider audience. We realize that somehow, we missed some clear statements in the text as to who the audience was, what the method for releasing results would be, and whether this would actually make things easier for new users, or whether this was directed at experts in ISSM who want to release professional quality results to a larger audience. We have accordingly corrected the introduction of the manuscript to make this abundantly clear. We would like to respectfully push back on the statement that the models presented in VESL are relatively simple. These are reseach-grade simulations, with resolutions and simulation times difficult to achieve without the use of a cluster. The SeaRISE experiment presented in Figure 4 was litterally taken from the ISSM SeaRISE runs carried out in 2014. This goes to a deeper misunderstanding that we tried to clarify in the manuscript, that originated from the savemodel routine description, in which both reviewers thought that we were caching the results. It is absolutely not the intent of this new API to do so, as we believe that having ISSM run live during a simulation is a key feature. We do not agree with the reviewer that this would be a waste of computational ressources, as this would be an exciting opportunity to showcase science capabiliities at their maximum potential, without degrading their quality, which is usually the first casualty of outreach and education. We again make this point clear in the text, and why we believe this is the way outreach should go.

So how accessible is this API for a non-specialist to use to set up their own domain, and to set up the web server, for example on the Amazon EC2 infrastructure? And what is the cost? This is not clear to me from the paper as it is written, and leaves me in doubt as to whether the API would simplify things for a less experienced user to set up their own domain, and even whether this is one of the intentions for the API.

This relates to the point raised before by the reviewer. The tool is not intended to facilitate use of ISSM per se, but its integration in a new environment that is web based. We are currently working on a full-fledged ISSM simulation portal on a webpage, but most of the time, this API will be used in conjunction with an already setup model from a scientist (from pre-existing Matlab or Python runs) to enable quicker dissemination of the results, and replication of the simulation on a webpage. This will not make things easier in terms of modeling, but it will make things easier on the scientist who will not need to be an expert in web design to on his own transfer his result and simulation engine to a webpage. Again, we have modified the introduction in this respect to make these points clear.

I think what I would like to see is a clearer statement of the purpose and advantages for different users, perhaps the information is there, but I got to the end of the paper not entirely the wiser.

See above.

Specific comments:

Lines 75-85: This numbered list is hard to follow because some of your points are long and contain full stops, I forgot what the list was about by the time I hit point 2). I suggest adding line breaks before each point, making them more like a bullet pointed list.

We actually took the reviewer's advice litterally, and went for an enumeration list, as the four points presented here are significant enough.

Line 79: Nothing should be considered as obvious, please remove the statement, or elaborate on the reasons!

Thank you for the advice, we have rephrased the statement accordingly to further explain why we believe HTML and JS are languages not used by the scientific community.

Lines 132 onwards: as with reviewer 1, I got a bit confused in this bit as to what was written in Matlab and what was in C++, please clarify.

The paragraph has been reworked (see equivalent remark from reviewer 1) to clearly differentiate the C++ core (used for large-scale model runs on the Amazon EC2 instances) and the JavaScript core (equivalent of the C++ core but translated into JS using emscripten) , used for computations local to the webpage that do not require parallel computing (small models).

Line 215: If the user wants to present the results in a different way, is it possible

for them to extract the data from the API, or do they have to use the inbuilt visualisation options?

This is a very interesting question. The results are provided directly by the call to the "solve" routine (see Listing 3) in engine.js. The user is therefore free to modify engine.js to use his own rendering engine directly. We have hinted at this aspect in the manuscript.

Line 239: remove the is after webpage

Done.

Manuscript prepared for Geosci. Model Dev.
with version 2015/04/24 7.83 Copernicus papers of the LaTeX class copernicus.cls.
Date: 27 January 2017

A JavaScript API for the Ice Sheet System Model (ISSM) 4.11: towards an online interactive model for the Cryosphere Community

Eric Larour[1], Daniel Cheng[2], Gilberto Perez[2], Justin Quinn[2], Mathieu Morlighem[3], Bao Duong[4], Lan Nguyen[5], Kit Petrie[1], Silva Harounian[6], Daria Halkides[7], and Wayne Hayes[2]

[1]Jet Propulsion Laboratory - California Institute of technology, 4800 Oak Grove Drive MS 300-323, Pasadena, CA 91109-8099, USA
[2]University of California, Irvine, Department of Information and Computer Sciences, Donald Bren Hall, Irvine, CA 92697-3100, USA
[3]University of California, Irvine, Department of Earth System Science, Croul Hall, Ivine, CA 92697-3100, USA
[4]Monoprice, Inc.11701 6th Street Rancho Cucamonga, CA 91730, USA.
[5]Hart, Inc., 1515 E Orangewoord ave Anaheim, CA 92805 USA
[6]Digitized Schematic Solutions, Address: 40 W. Cochran st. Suite: 212 Simi Valley CA 93065, California, USA
[7] Earth and Space Research, 2101 Fourth Ave., Suite 1310, Seattle, WA 98121, USA.

Correspondence to: Eric Larour (eric.larour@jpl.nasa.gov)

Abstract.

Earth System Models (ESMs) are becoming increasingly complex, requiring extensive knowledge and experience to deploy and use in an efficient manner. They run on high-performance architectures that are significantly different from the everyday environments that scientists use to pre and post-process results (i.e. MATLAB, Python). This results in models that are hard to use for non specialists, and that are increasingly specific in their application. It also makes them relatively inaccessible to the wider science community, not to mention to the general public. Here, we present a new software/model paradigm that attempts to bridge the gap between the science community and the complexity of ESMs, by developing a new JavaScript Application Program Interface (API) for the Ice Sheet System Model (ISSM). The aforementioned API allows Cryosphere Scientists to run ISSM on the client-side of a webpage, within the JavaScript environment. When combined with a Web server running ISSM (using a Python API), it enables the serving of ISSM computations in an easy and straightforward way. The deep integration and similarities between all the APIs in ISSM (MATLAB, Python, and now JavaScript) significantly shortens and simplifies the turnaround of state-of-the-art science runs and their use by the larger community. We demonstrate our approach via a new Virtual Earth System Laboratory (VESL) Web site.

1 Introduction

Earth System Models (ESMs) across the Earth science community have become increasingly sophisticated, enabling more accurate simulations and projections of the Earth's climate as well as the state of the atmosphere, ocean, land, ice, and biosphere. As demonstrated by the Coupled Model Intercomparison Project 5 (CMIP-5, Taylor et al., 2009, 2012) and its new iteration (CMIP-6, Eyring et al., 2016) of the World Climate Research Programme (WCRP), the multiplicity of ESMs, and the complexity of the physics they capture, is significant. The description of the outputs for CMIP-5 runs is 133 pages long by itself, showing the complexity and comprehensive nature of the processes modeled in the ESMs that participated in the project. Any one of these models is massive both in terms of the number of lines of code, but also in terms of structure and modularity (or lack thereof). GEOS-5 for example (Molod et al., 2015), one of the Atmosphere and Ocean General Circulation Models (AOGCMs) that participated in CMIP-5, is made of 600,000 lines of Fortran code, comprising 88 physical modules (as of Jan 2016). This is fairly representative of the complexity of ESMs nowadays, and of the multiplicity of physical processes necessary to realistically model the evolution of the whole Earth System.

The above described complexity results in serious issues regarding the way simulations are run. For example, what we generally define as pre-processing and post-processing phases are increasingly different from the computational phase itself. The computational core is usually written in C or Fortran, which easily supports parallelism and High Performance Computing (HPC). However, in the pre-processing phase, where datasets are processed into a binary file used by the computational core, or in the post-processing phase, where simulation results are visualized, scientific environments such as MATLAB or Python are increasingly relied upon. This results in additional complexity to manage different environments: scientists are well-acquainted with the difficulties of porting their software to HPC instances, while struggling to process the data inputs and results on local workstations where data upload/download can be a limiting factor, hard drive memory requirements substantial, and problems due to the use of different APIs significant (MATLAB, Python, and IDL, among others).

Another complexity originating from the wide variety of physical processes represented in ESMs is the difficulty in initializing a computational run. For example, in the Ice Sheet System Model (ISSM, Larour et al., 2012), one of the land ice components of GEOS-5, developed at the National Aeronautics and Space Administration (NASA) Jet Propulsion Laboratory (JPL), in collaboration with University of California, Irvine (UCI), the initialization setup for the Greenland Ice Sheet (GIS) transient simulations from 1850 to present day amounts to 3,000 lines of MATLAB code. This comprises model setup, data interpolation onto an ISSM compatible mesh, solution parameterizations, and initialization strategies, among other things. This simulation, part of the Ice Sheet Modeling Intercomparison Project 6 (ISMIP-6 Nowicki et al., 2016) that accounts for ice sheets in CMIP-6, is a fairly representative example of some of the most advanced simulations that can be run with

an Ice Sheet Model (ISM). Such simulations cannot easily be systematized and need to be tailored specifically for each ice sheet they are applied to.

One of the approaches that could mitigate some of the issues discussed above involves the development of computational frameworks capable of serving ESM simulations. This type of solution involves running simulations that already include pre and post-processing phases (i.e.where the model setup has already been carried out or is carried out by the server itself by uploading key datasets) and in which the user is allowed to control only a few, key parameters. Similarly, once the computation is carried out on the server-side, the results are post-processed automatically, and only significant results are provided to the user. This type of approach has already been explored, for example, in areas relating to serving of large datasets, such as the NASA Earth Observing System Data and Information System (EOSDIS) EarthData server, which provides a portal with integrated processing capabilities for large scale datasets collected by NASA missions. However, fewer examples of this kind of approach are available that serve simulation results, and to our knowledge, no comprehensive ESM, nor module thereof, has ever been integrated into a server solution capable of delivering ESM computations on the fly. The reason for this is simple: the complexity of the physics involved is significant, reconciling pre/post processing phases and simulation cores is inherently difficult, and basing a simulation framework on server technologies represents a significant software development challenge.

Specifically, the bottlenecks that preclude deeper integration of ESMs within server infrastructures include:

1. Bridging the gap between ESM formulations of the physical cores and Web technologies such as HyperText Markup Language (HTML, World Wide Web Consortium, 1997) and JavaScript (ECMA International, 2016), which are not scientifically oriented languages and are thus not inherently used by Earth scientists. Because ESMs are not natively integrated into Web technologies, it renders the link between server infrastructures and simulation engines difficult

2. The significant turnaround between generation and serving of simulations. This lag is due to the fact that these two processes are inherently different in the way they are designed and, moreover, are usually considered to be completely separate phases of what should, essentially, be the same process.

3. The distributed nature of Web simulations. Every step of an ESM run can be considered a separate, logical component. For example, post-processing of a simulation may be done on a different machine than the one that initially generated it.

4. The lack of existing integrated frameworks wherein simulations, pre and post-processing, and the serving of the data and/or simulation results all occur within the same architecture.

Here, we present a new approach applied to the ISSM framework, a land ice model of significant size and complexity, to serve simulations relating to the evolution of polar ice sheets. Here, by serv-

ing, we imply providing a way to run simulations interactively within a Web environment, without any of the results ever being cached. Our solution is based on a new JavaScript API for the ISSM framework itself, allowing it to be fully integrated within an HTML webpage (described in Section 2) and to run local to the webpage. For models of larger size, we also show how we leverage the existing ISSM Python API to run a web server (based on Apache and the FastCGI module) that can run faster parallel computations, and to which the webpage client can upload model inputs and download computation along with pre and post-processing results directly (Section 3). This new approach allows for a quick turnaround between running simulations and porting such simulations to a webpage interface for access to the wider science community (Section 4). We execute this approach (Section 5) within the newly-designed Virtual Earth System Laboratory (VESL), demonstrating how we can provide access to cryosphere-related simulations to the science community, and to the wider public in general, thereby easily providing access to the wide array of modular physics embedded in ISSM. We conclude with a discussion of the potential of this new approach to both facilitate a wider use of ESMs by scientists of varied disciplines and to shorten the gap between science simulations and public outreach.

2 ISSM JavaScript API

Most ISMs are written in Fortran, C, or C++, for reasons related to computational efficiency and to the ease of integration within HPC environments using parallel libraries, such as Message Passing Interface (MPI) via OpenMP (Gropp et al., 1996; Gropp and Lusk, 1996; OpenMP Architecture Review Board, 2015). However, many simpler models exist that rely on different APIs, such as the MATLAB code described in MacAyeal (1993) or the Excel-based Greenland and Antarctica Ice Sheet Model designed for educational purposes (GRANTISM, Pattyn, 2005). These models have in common the desire to rely on a simple code base, and to reduce/optimize the set of physics captured in the code, in order to make it more accessible. Our approach here, however, is to facilitate accessibility without sacrificing the complexity and full-set of features of ISSM by implementing a brand new API using the JavaScript language. The goal is to be able to integrate ISSM within Web-based solutions, relying on JavaScript as a language that enables control of the behavior of an HTML webpage. In addition, by making the JavaScript API similar in all possible aspects to the existing MATLAB and Python ISSM APIs, model runs and simulations can be transferred easily to the Web, furthering our objective of disseminating ISSM to the larger scientific community and, possibly, the general public through Web interfaces. It is to be noted that the new API being of equivalent complexity (to capture the full range of physics) to the MATLAB or Python APIs, users that want to use this API should be fully knowledgeable with using ISSM in MATLAB or Python already. This means that the new API does not make use of ISSM easier in terms of learning curve, but makes it more flexible in terms of being deployed to the Web.

The basis for representing a model in ISSM is a series of classes (mesh, mask, geometry, settings, toolkits, etc.) that are carried into a global `model` class. The first task was, therefore, to translate all ISSM classes from MATLAB and Python into JavaScript. Fig. 1 shows an example of such a translation for the `mesh2d` class (used to represent a 2D mesh triangulation comprising a list of vertex coordinates `x`,`y` of size `numberofvertices` with corresponding `lat`,`long` coordinates, a list of triangle indices called `elements` (of size `numberofelements`), and a projection code using an EPSG Geodetic Parameter Dataset). The constructors are very similar, and there is a one-to-one correspondence between the mesh2d methods in both APIs. The example of the `marshall` routine (which collects all the mesh info onto a binary buffer that will be sent to the ISSM C++ core) shows the similarity between both codes, with differences in the syntax reduced to a bare minimum. This equivalence is essential in preserving all of the physics captured in each class of ISSM, and could only be achieved because MATLAB, Python, and JavaScript are similar in their syntax and grammar.

In a standard modeling analysis, scientists will develop their models and run within the MATLAB (or Python) environment. Usually, outreach of the results will be done separately, in a different web based environment, leading to inefficiencies and potential loss of information/accuracy between the science analysis and the outreach itself. To remedy this issue, it is very convenient to provide an efficient way to transfer a model directly from MATLAB to the JavaScript environment, where it will be loaded easily using a standard 'include' statement. This is implemented through the `savemodel` routine for each subclass of the `model` class. As shown in Fig. 1 for the `mesh2d` class implementation, the `savemodel` routine allows users to write the MATLAB model to a JavaScript file. This allows users to run simulations in MATLAB using ISSM, and, once the simulations are over, to save the MATLAB defined model into a JavaScript equivalent file. This routine, which closely matches the constructor, is the key to shortening the transition time between the setup of an ISSM simulation and its transition into a webpage environment. The fact that all of the information of a given class is identical in both APIs demonstrates the comprehensiveness of the new JavaScript implementation of ISSM, and that it achieves its goal of replicating ISSM within a webpage environment.

In a standard model run, MATLAB classes (or Python) are used to setup the model, but the computations themselves are carried out in C++. This C++ code is present at several levels: 1) For each pre and post-processing module (or, wrapper) that requires significant computational power, such as interpolation routines that transfer information between gridded dataset and unstructured Finite Element Modeling (FEM) meshes typical of ISSM; and 2) For each of the computations pertaining to ice flow itself (the physical engine in ISSM), which we refer to as the ISSM core. For pre and post-processing modules, the computations are assumed local to the workstation. For the ISSM core itself, parallelization is inherent (using the MPI libraries), and this core usually runs on a parallel cluster.

When we look at this configuration and try and transfer this paradigm to a webpage environment, we are however faced with two issues: 1) C++ code cannot be run native to a webpage easily and 2) parallelism is not yet implemented in browsers, and would anyway result in heavy taxation of CPU ressources (on local workstations/laptops/tablets), which is not practical. We therefore approached this issue in two ways: 1) we translated the entire C++ code (both modules/wrappers and the ISSM core itself) into JavaScript for model runs that are small enough to be run locally; and 2) for models that are too large to run local, we implemented a way of uploading (using the JavaScript classes) a model to a web server on the Amazon EC2 cloud, where computations are carried out and returned to the JavaScript client once completed. The latter approach is described in the next section. We here further describe the translation of the C++ modules and ISSM core into JavaScript code. This translation was carried out using the Emscripten compiler (Zakai, 2011) . This compiler enables translation of C++ code directly into JavaScript, with computational efficiencies that are within an order of magnitude of the translated C++ code. Listing 1 shows how Emscripten was integrated within the existing Makefile structure of ISSM. All the pre and post-processing wrappers (TriMesh, NodeConnectivity, ContourToMesh, ElementConnectivity, InterpFromMeshToMesh2d, IssmConfig, EnumToString, and StringToEnum) as well as the ISSM core itself (issm) are compiled into JavaScript executables using the C++ files and a set of Emscripten related flags (described in the IssmModule_CXXFLAGS variable). This Makefile is similar to its MATLAB and Python counterparts, with the exception of the issm core, which is compiled as a JavaScript module instead of a C++ executable. This Makefile is integrated within Autotools (Vaughan et al., 2000), enabling for quick activation of the compilation using a simple "`--with-javascript`" option during the configuration phase of the ISSM software.

The JavaScript modules and ISSM core are continuously tested against regression tests, similar to the MATLAB and Python APIs (Larour et al., 2012). The integration framework for the tests relies on Jenkins, an open-source automation server (Jenkins, 2016), which provides continuous integration and delivery of validated ISSM code. The ISSM Jenkins webpage is available at https://ross.ics.uci.edu:8080/, where the entire validation suite is in the process of being transferred to JavaScript. This ensures that continuous development impacts all of the APIs in ISSM in a similar fashion without imparting delays to the JavaScript API (due to the fact that it would be used by a smaller base of ISSM users).

3 HTTP/Python Server

Using the JavaScript API, it is possible to run a full-fledged simulation using any of the physical modules described in Larour et al. (2012). However, to our knowledge, Emscripten does not yet allow computations in parallel within a browser. This limits the range of model sizes and mesh resolution to a level that compromises large-scale simulations. In these cases, our approach was to

rely on the cloud computing capabilities of ISSM, as described in Larour and Schlegel (2016), and to host a Web server that would deliver ISSM computations to any client running the ISSM JavaScript API. This server relies on the Python API of ISSM to carry out computations ranging from tens to hundreds of thousands of degrees of freedom, allowing continental-scale simulations. The server is fully-elastic and scalable, and relies on the Amazon EC2 infrastructure (Inc, 2008), and can spin-up Compute Optimized CC4.8x large instances (up to 64 threads of computational power) on demand, making it a robust solution for serving computations. Refer to Larour and Schlegel (2016) for more details on this part of the architecture.

In terms of server configuration itself, our approach was to rely on the Python API of ISSM to leverage the FastCGI Web interface, described in Market (1996), on an Apache server. This allows requests coming into the Apache server from the client-side to be routed directly to a Python script. The Web client, running ISSM embedded inside JavaScript, can therefore upload a marshalled binary input file (created by the call to the marshall routine of each model class, as described in Fig. 1) to the EC2 instance Apache server, which then routes it to the Python script that launches the parallel job.

Fig. 2 describes this process schematically, and compares it to what happens in more classic simulations relying on MATLAB and an HPC infrastructure, such as a cluster. The fundamental differences between the traditional simulation paradigm and our new solution are: 1) The client architecture, which runs either MATLAB or an HTML webpage with JavaScript; 2) The upload/download of binary input files, which is done either through an SSH copy call or an XMLHttpRequest, respective to the aforementioned client architectures; and 3) The launching of a given computation, which is handled via a queuing system on the head node or a FastCGI-relayed Python call on an EC2 instance, again, respective to the client architecture. In terms of parallel computations, ISSM executables are run using an MPI call in both cases. The strong similarity between both architectures was purposefully designed so as to limit the amount of repeated code, and to ensure the robustness of the computations themselves, which are transparent to the API they rely upon.

4 All-In-One Design/Simulations

Listing 2 shows a typical model setup for a simulation in ISSM relying on the MATLAB API. The steps include loading a model (or generating one using a mesher), modifying a certain input parameter, setting up a cluster class (pointing to the parallel cluster) and calling the solve routine. Once the results are carried out/downloaded, plotmodel is run to visualize them.

An additional step can be carried out once a given MATLAB ISSM model has been built, wherein the model is saved into a JavaScript file (md.js) in some webpage directory. This model can then be used (as shown in Listing. 3) to run the exact same setup and simulation as is done with MATLAB, but on the client's machine. The HTML code for this simulation is typical of a webpage, and in-

cludes: 1) Standard HTML markup (i.e. W3C-compliant html, head, and body objects); 2) Include statements for the ISSM binaries created by Emscripten, the model itself (md.js), and a sort of front-end controller (engine.js, which controls the display of and interaction with the simulation on the webpage); and 3) HTML elements such as a canvas where the results will be plotted (similar to the figure statement in MATLAB), a second canvas for the color bar, and a button element to launch the simulation. The listing for engine.js shows how similar the MATLAB and JavaScript setup are. Upon loading, if the RUN button is clicked, the value of a slider (the model input of interest) is retrieved and then SolveGlacier called. The SolveGlacier() routine modifies the input parameter, sets up the cluster class (pointing to the EC2 server), and calls the solve routine. After computations are carried out and downloaded, a callback function PlotGlacier is triggered, which plots the model results onto the aforementioned HTML canvas elements. If users do not want to rely on this particular routine for plotting, they can instead provide their own callback routine to plot using their own rendering engine.

Fig. 3 shows an example of such a webpage hosting a simulation for the Columbia Glacier, Alaska. In this particular example, the model input that is modified is the surface mass balance (SMB). This parameter measures the amount of precipitation (in snow or water) at the surface of the ice, minus runoff of water from melting and evaporation. This parameter is essential in controling the input of mass to the glacier. Once this input is modified, we can measure its impact on the response of the glacier (the ice flow) through time. This response is a complex interplay between mass transport processes and the stress equilibrium of the ice. The result is a new flow regime (speed), which ISSM can compute and which can be visualized through a time evolution of the speed at the surface of the ice.

Here, the webpage is part of VESL, where the JavaScript API of ISSM was leveraged along with the HTTP/Python Server architecture described previously to showcase the capabilities of ISSM to serve computations on the fly and to visualize them instantly (Larour et al., 2016). The simulations within VESL are all simulations that were carried out using ISSM for scientific publications. By adding a savemodeljs step at the end of the MATLAB simulation workflow, we were able to transfer the model used for the simulations from the MATLAB environment onto the webpage. Once that was done, we replicated a workflow similar to the MATLAB workflow in the engine.js code. With this approach, it is possible to deploy a simulation like the one described above on a Web platform with significantly shortened turnaround and using the exact same capabilities as the initial MATLAB solution itself. This breakthrough is only possible because of the duplication of the entire architecture: again, by making JavaScript code that is logically equivalent to our MATLAB or Python constructs and by mapping the whole workflow described in Fig. 2 from MATLAB/HPC infrastructures to HTML/JavaScript/EC2. Our methodology paves the way to leveraging Web technologies and cloud computing to host large-scale simulations of modeling engines such as ISSM, all without loss of the physical representation of processes nor scalability.

5 Examples

Fig. 3 and Fig. 4 show examples of simulations that rely on the ISSM JavaScript API, and that are hosted on the VESL Web site (Larour et al., 2016). VESL's purpose is to twofold: to showcase simulations that demonstrate ISSM capabilities, and to demonstrate the capabilities of our new Web-based modeling solution to the wider scientific community and general public. Several simulations are hosted, leveraging the large set of capabilities in ISSM.

The first simulations pertain to the simulation of glacier flow, mainly from work on Haig Glacier (Adhikari and Marshall, 2011) and Columbia Glacier (Gardner, Fahnestock, Larour, pers. comm.). Fig. 3 shows the Columbia Glacier webpage, where SMB variations can be specified, with ISSM then computing the resulting impact on the glacier evolution over a period of 10 years. This simulation includes all the physical processes that control the evolution of a glacier, and is fully representative of the complex physics required in the analysis.

The second set of simulations pertain to ice sheet modeling in Antarctica and Greenland. Fig. 4 shows the webpage corresponding to the friction SeaRISE (Bindschadler et al., 2013) experiment over the entire Greenland ice sheet. In this simulation, the user can decrease the friction at the ice/bedrock interface under the ice sheet and compute the resulting changes in steady-state velocity at the surface. The model is fairly high resolution (12,000 elements), which allows for computationas that are physically representative.

The third set of simulations pertains to Sea-Level Rise (SLR) modeling, relying on the ISSM-SESAW module (Adhikari et al., 2016) to compute gravitationally consistent sea-level and geodetic signatures caused by cryosphere and climate-driven mass change. Presently, two sets of simulations demonstrate: 1) Eustatic SLR and its impact on coastline migration in the USA; and 2) SLR from eustatic, gravity, and elastic deformation on a global scale, wherein users can turn off specific sets of SLR physics to understand the impact of gravitation on redistribution of SLR around the world and the impact of local elastic deformation of the Earth lithosphere.

Finally, a fourth set of simulations pertains to Solid Earth deformation, using the ISSM-GIA (Adhikari and Marshall, 2011) module that captures Glacial Isostatic Adjustment (GIA) from ice-sheet loading. It should be noted that this section is a work in progress.

One potential future section may feature recent work by the ISSM team involving the application of ISSM to other planets (namely, Mars' ice caps). Given the relatively quick turnaround between ISSM simulations and their porting to the Web using the ISSM JavaScript API, our hope is that VESL will become a forum for cryosphere scientists to discuss ice sheet related science. In addition, by enabling simplified interfaces on existing simulations that resulted in scientific publications, we believe the general public might gain increased interest in this type of approach to better understand the complexities of science for the Earth system as a whole.

6 Conclusions

We developed a fully-functional JavaScript API for the Ice Sheet System Model (ISSM), which allows cryosphere scientists to carry out ice flow simulations within a Web environment. This API gives access to the entire spectrum of physical processes captured by ISSM without compromising its complexity and richness. For simulations requiring parallel computing, the JavaScript API can be leveraged against a computational server hosted on a cloud instance (such as Amazon EC2) to deliver high-performance, large-scale, and high-fidelity simulations back to the Web client. This new set of capabilities enables hosting of high-end simulations on the NASA/JPL ESL, effectively solving a fundamental challenge of ESMs: delivering accessible, high-performance simulations in a timely manner is historically and inherently difficult. We believe that our approach paves the way for the efficient deployment of feature-rich ESM's, a quick turnaround between scientific work and corresponding publications, and outreach not only the science community but also to the general public.

7 Code Availability

The ISSM code and its JS components are available at http://issm.jpl.nasa.gov. The instructions for the compilation of ISSM in JS mode is presented in the supplement attached to this manuscript.

Acknowledgements. This work was performed at the Jet Propulsion Laboratory (JPL), California Institute of Technology, and the Department of Earth System Science at the University of California, Irvine (UCI) under a contract with the National Aeronautics and Space Administration (NASA) and funded by the Cryospheric Sciences Program. Resources supporting the numerical simulations were provided by the NASA High-End Computing (HEC) Program through the NASA Advanced Supercomputing (NAS) Division at Ames Research Center, and by Cryospheric Program Management for the Amazon EC2 instances hosting the Virtual Earth System Laboratory. We would like to thank Dr. Alex Gardner from JPL and Dr. Mark Fahnestock from the University of Alaska, Fairbanks for the datasets used in the Columbia Glacier model setup of the Virtual Earth System Laboratory. We would also like to thank Dr. Daisy F. Sang for the supervision of students from CalPoly Pomona who participated in this project over the past 5 years.

Listing 1. Makefile for Javascript Emscripten compilation of ISSM.

```
EXEEXT=.js

js_scripts = ${ISSM_DIR}/src/wrappers/TriMesh/TriMesh.js   \
${ISSM_DIR}/src/wrappers/NodeConnectivity/NodeConnectivity.js\
${ISSM_DIR}/src/wrappers/ContourToMesh/ContourToMesh.js\
${ISSM_DIR}/src/wrappers/ElementConnectivity/ElementConnectivity.js\
${ISSM_DIR}/src/wrappers/InterpFromMeshToMesh2d/InterpFromMeshToMesh2d.js\
${ISSM_DIR}/src/wrappers/IssmConfig/IssmConfig.js\
```

```
    ${ISSM_DIR}/src/wrappers/EnumToString/EnumToString.js\
    ${ISSM_DIR}/src/wrappers/StringToEnum/StringToEnum.js\
    ${ISSM_DIR}/src/wrappers/Issm/issm.js

340 bin_SCRIPTS  = issm-prebin.js
    bin_PROGRAMS = IssmModule

    issm-prebin.js: ${js_scripts}
      cat ${js_scripts} > issm-prebin.js
345
    IssmModule_SOURCES = ../TriMesh/TriMesh.cpp \
                ../NodeConnectivity/NodeConnectivity.cpp\
                ../ContourToMesh/ContourToMesh.cpp\
                ../ElementConnectivity/ElementConnectivity.cpp\
350             ../InterpFromMeshToMesh2d/InterpFromMeshToMesh2d.cpp\
                ../IssmConfig/IssmConfig.cpp\
                ../EnumToString/EnumToString.cpp\
                ../StringToEnum/StringToEnum.cpp\
                ../Issm/issm.cpp
355
    IssmModule_CXXFLAGS= -fPIC -D_DO_NOT_LOAD_GLOBALS_ --memory-init-file 0 \
    $(AM_CXXFLAGS) $(CXXFLAGS) $(CXXOPTFLAGS) $(COPTFLAGS) \
    -s EXPORTED_FUNCTIONS="['_TriMeshModule','_NodeConnectivityModule',\
    '_ContourToMeshModule','_ElementConnectivityModule',\
360 '_InterpFromMeshToMesh2dModule','_IssmConfigModule','_EnumToStringModule'\
    ,'_StringToEnumModule','_IssmModule']" -s DISABLE_EXCEPTION_CATCHING=0 \
    -s ALLOW_MEMORY_GROWTH=1 -s INVOKE_RUN=0

    IssmModule_LDADD = ${deps} $(TRIANGLELIB)  $(GSLLIB)
```

```
%MESH2D class definition
classdef mesh2d
    properties (SetAccess=public)
        x                           = NaN;
        y                           = NaN;
        elements                    = NaN;
        numberofelements            = 0;
        numberofvertices            = 0;
        numberofedges               = 0;
        lat                         = NaN;
        long                        = NaN;
        epsg                        = 0;
    end
    methods
+-- 18 lines: function self = mesh2d(varargin) % ----------------------------------------------------------------
+--  9 lines: function self = setdefaultparameters(self) % ---------------------------------------------------
+-- 19 lines: function md = checkconsistency(self,md,solution,analyses) % ----------------------------------------
function marshall(self,md,fid) % {{{
WriteData(fid,'enum',DomainTypeEnum(),'data',StringToEnum(['Domain' domaintype(self)]),'format','Integer');
WriteData(fid,'enum',DomainDimensionEnum(),'data',dimension(self),'format','Integer');
WriteData(fid,'enum',MeshElementtypeEnum(),'data',StringToEnum(elementtype(self)),'format','Integer');
WriteData(fid,'object',self,'class','mesh','fieldname','x','format','DoubleMat','mattype',1);
WriteData(fid,'object',self,'class','mesh','fieldname','y','format','DoubleMat','mattype',1);
WriteData(fid,'enum',MeshZEnum(),'data',zeros(self.numberofvertices,1),'format','DoubleMat','mattype',1);
WriteData(fid,'object',self,'class','mesh','fieldname','elements','format','DoubleMat','mattype',2);
WriteData(fid,'object',self,'class','mesh','fieldname','numberofelements','format','Integer');
WriteData(fid,'object',self,'class','mesh','fieldname','numberofvertices','format','Integer');
end % }}}
+--  3 lines: function t = domaintype(self) % ----------------------------------------------------------------
+--  3 lines: function d = dimension(self) % -----------------------------------------------------------------
+--  3 lines: function s = elementtype(self) % ---------------------------------------------------------------
function savemodeljs(self,fid,modelname) % {{{
writejs1Darray(fid,[modelname '.mesh.x'],self.x);
writejs1Darray(fid,[modelname '.mesh.y'],self.y);
writejs2Darray(fid,[modelname '.mesh.elements'],self.elements);
writejsdouble(fid,[modelname '.mesh.numberofelements'],self.numberofelements);
writejsdouble(fid,[modelname '.mesh.numberofvertices'],self.numberofvertices);
writejsdouble(fid,[modelname '.mesh.numberofedges'],self.numberofedges);
writejs1Darray(fid,[modelname '.mesh.lat'],self.lat);
writejs1Darray(fid,[modelname '.mesh.long'],self.long);
writejsdouble(fid,[modelname '.mesh.epsg'],self.epsg);
end % }}}
end
end
```

```
//MESH2D class definition
function mesh2d () {
    //methods
+-- 10 lines: this.setdefaultparameters = function (){ ----------------------------------------------------------
+--  3 lines: this.classname = function () { ----------------------------------------------------------------
+--  3 lines: this.domaintype=function (){ ------------------------------------------------------------------
+--  3 lines: this.dimension = function () { ----------------------------------------------------------------
+--  3 lines: this.elementtype = function() {---------------------------------------------------------------
this.marshall=function(md,fid) { //{{{
WriteData(fid,'enum',DomainTypeEnum(),'data',StringToEnum('Domain' + this.domaintype()),'format','Integer');
WriteData(fid,'enum',DomainDimensionEnum(),'data',this.dimension(),'format','Integer');
WriteData(fid,'enum',MeshElementtypeEnum(),'data',StringToEnum(this.elementtype()),'format','Integer');
WriteData(fid,'object',this,'class','mesh','fieldname','x','format','DoubleMat','mattype',1);
WriteData(fid,'object',this,'class','mesh','fieldname','y','format','DoubleMat','mattype',1);
WriteData(fid,'enum',MeshZEnum(),'data',NewArrayFill(this.numberofvertices,0),'format','DoubleMat','mattype',1)
WriteData(fid,'object',this,'class','mesh','fieldname','elements','format','DoubleMat','mattype',2);
WriteData(fid,'object',this,'class','mesh','fieldname','numberofelements','format','Integer');
WriteData(fid,'object',this,'class','mesh','fieldname','numberofvertices','format','Integer');
}//}}}
+-- 12 lines: this.fix=function() { ------------------------------------------------------------------------------
    //properties
    // {{{
        this.x                          = NaN;
        this.y                          = NaN;
        this.elements                   = NaN;
        this.numberofelements           = 0;
        this.numberofvertices           = 0;
        this.numberofedges              = 0;

        this.lat                        = NaN;
        this.long                       = NaN;
        this.epsg                       = 0;

        this.setdefaultparameters();
        //}}}
}
~
```

Figure 1. Line by line comparison of the code behind the mesh2d class, within the MATLAB ISSM API (upper frame) and the JavaScript ISSM API (lower frame). Routines followed by a dashed line have been folded for ease of reading.

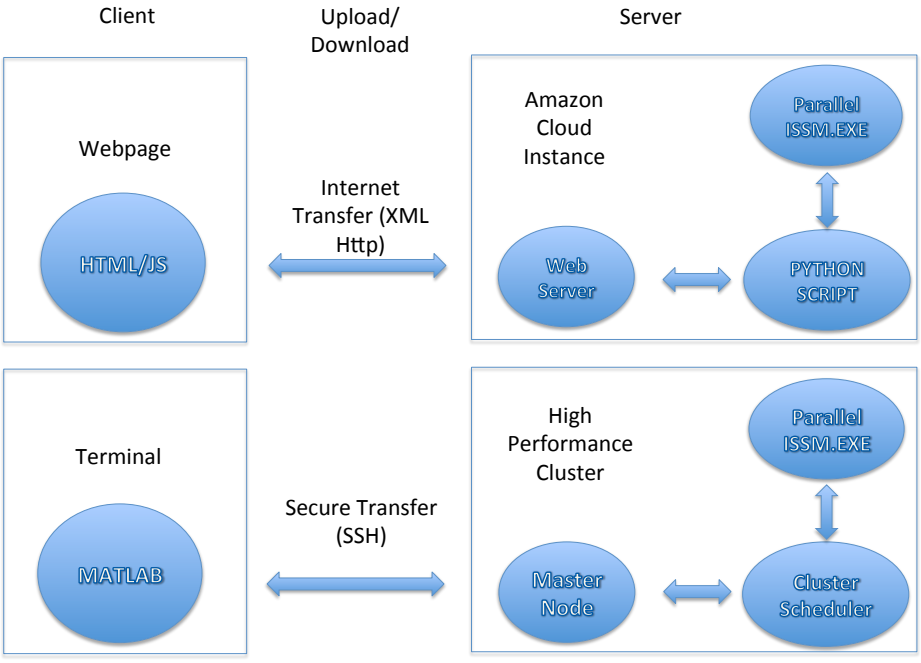

Figure 2. Similarities between a standard ISSM run from a terminal running MATLAB and connected to a high-performance cluster (HPC) and a web based ISSM run from a Webpage runing JavaScript, connected to a Web server runing on an Amazon EC2 Cloud instance. In the first case (lower frames) a MATLAB instance running on a local workstation terminal running marshalls an input binary file, which is then uploaded (using an ssh call) to a master node on a cluster. The binary file is then queued into the system (using a qsub command from a scheduler, for example). The parallel runs are then carried out using the ISSM executable and an MPI-compatible environment. In the second case, a browser client makes an XMLHttpRequest and uploads a Binary Large Object (the exact same binary file MATLAB would upload), which is received by an HTTP server (e.g. Apache) running on an Amazon EC2 compute-optimized instance. The HTTP server then uses a FastCGI module to interface to a Python wrapper, which automatically triggers a system call to the MPI environment running the ISSM executable. In both cases, an output binary file is created by the ISSM executable, which is then shipped back to the MATLAB instance or the client's Web browser.

---

365 ;

Listing 2. MATLAB code for a typical simulation of the Virtual Earth System Laboratory (VESL).

```
% Load Model:
md=loadmodel('Models/md.mat');

% Solve:
```

```
md.smb.mass_balance= smb_initial;
for i=1:md.mesh.numberofvertices,
  md.smb.mass_balance(i) = md.smb.mass_balance(i)+ smbvalue;
end

md.cluster=generic('name','localhost','np',8);
md=solve(md,TransientSolutionEnum());

%Plot Model:
vel=md.results.TransientSolution(1).Vel;
plotmodel(md,'data',vel,'log',10,'figure',1,'colorbar','on',...
      'overlay','on','images','radar.png');

% Export to JS model:
md.savemodeljs('md',websiteroot);
```

Listing 3. Equivalent (see Listing. 2) Hmtl/Javascript code for a typical simulation within the Virtual Earth System Laboratory. Prototype webpage.

```
<html>
  <script type="text/javascript" src="./bin/issm-binaries.js"></script>
  <script type="text/javascript" src="./src/engine.js"></script>
  <script type="text/javascript" src="./js/md.js"></script>

  <body data-spy="scroll" data-target="nav" onload="engine();">

    <canvas id="columbia"></canvas>

    <div class="bordered margin-8 padding-8">
      <canvas id="columbia-colorbar" class="colorbar-v"> </canvas>
    </div>

    <div id="columbia-run" class="bordered margin-8 padding-8">
      <button type="button" class="interactive run-button"
        onclick="SolveGlacier()"> RUN </button>
    </div>
  </body>
</html>

function engine(){
  PlotGlacier();
  slider('value',0,'callback',function(value){smbvalue=value;},
```

```
  'name','columbia', 'min',-5,'max',+5,'message',['SMB anomaly:','m/a'],
  'step',.1, 'slidersdiv','columbia-sliders');
}

function SolveGlacier(){
  md.smb.mass_balance= smb_initial.slice(0);
  for (var i=0;i<md.mesh.numberofvertices;i++){
    md.smb.mass_balance[i] += smbvalue;
  }
  md.cluster=new generic('url',server + '/fastcgi/issm_solve.py','np',8);
  md=solve(md,TransientSolutionEnum(),'checkconsistency','no',
  'callback',PlotGlacier);
}

function PlotGlacier(){
  plotmodel(md,'data',md.results[0]['Vel'],'log',10,'canvasid#all','columbia',
    'colorbar','on', 'colorbarcanvasid','columbia-colorbar',
    'overlay','on','image','./images/radar.png');
}
```

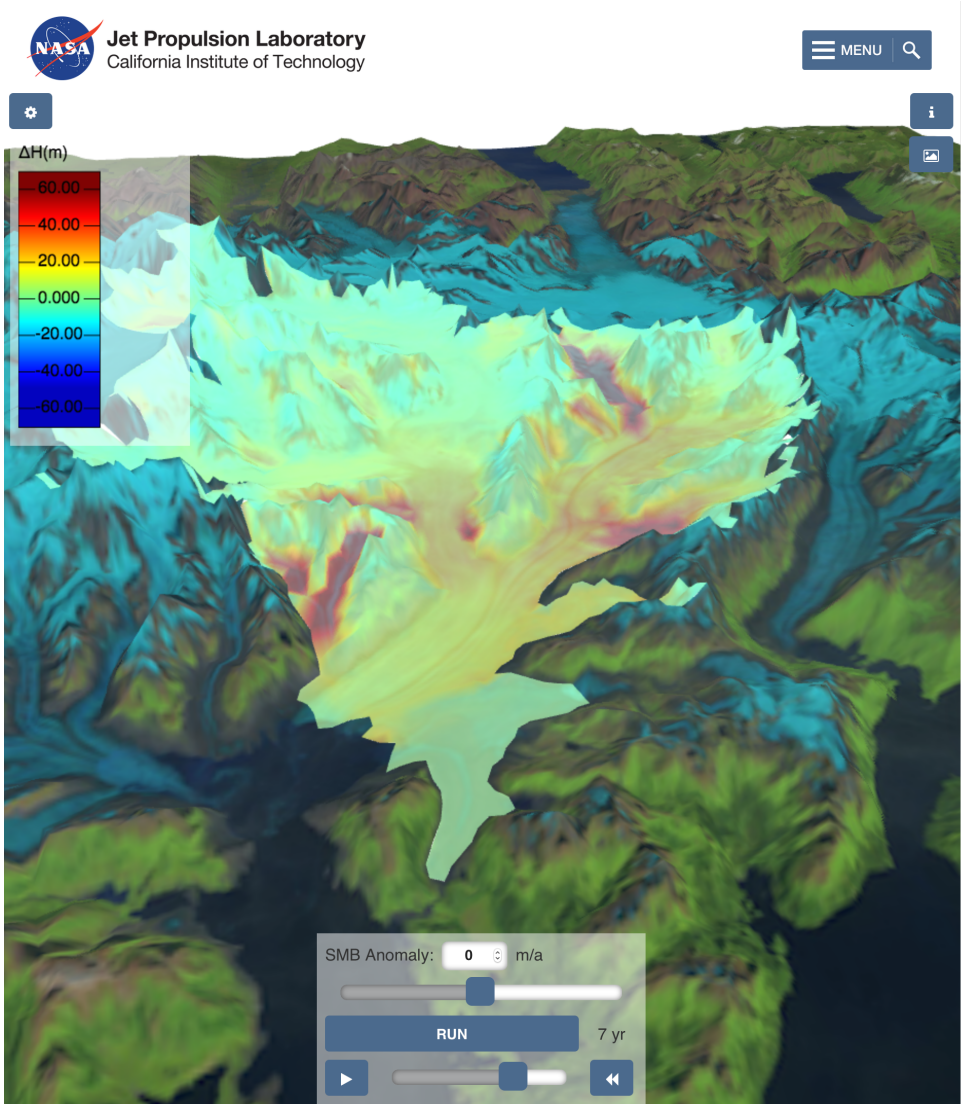

Figure 3. Columbia Glacier ISSM simulation on the Virtual Earth System Laboratory (http://issm.jpl.nasa.gov/earthsystemlaboratorynew). This particular simulation allows for the introduction of user-driven SMB anomalies (using a slider ranging from -5 to +5 m/a) on the transient ice flow of Columbia Glacier. The computations (upon clicking of the RUN button) are carried out on the ISSM computational server (where the model inputs are uploaded, and from which the results are downloaded locally to the client's Web browser). The transient results are displayed as a movie, which can be controlled via user interface (UI) controls. The interactive rendering of the velocity and thickness fields is done in 3D (or 2D, upon clicking of a toggle button) using the ISSM WebGL rendering engine. The results are overlaid on a semi-transparent topographical rendering of the SRTM DEM, and a background geotiff image from Gardner et al (pers. comm.). Model information can be displayed by clicking the info button, allowing for extensive information on the model setup and the datasets used to constrain the simulation.

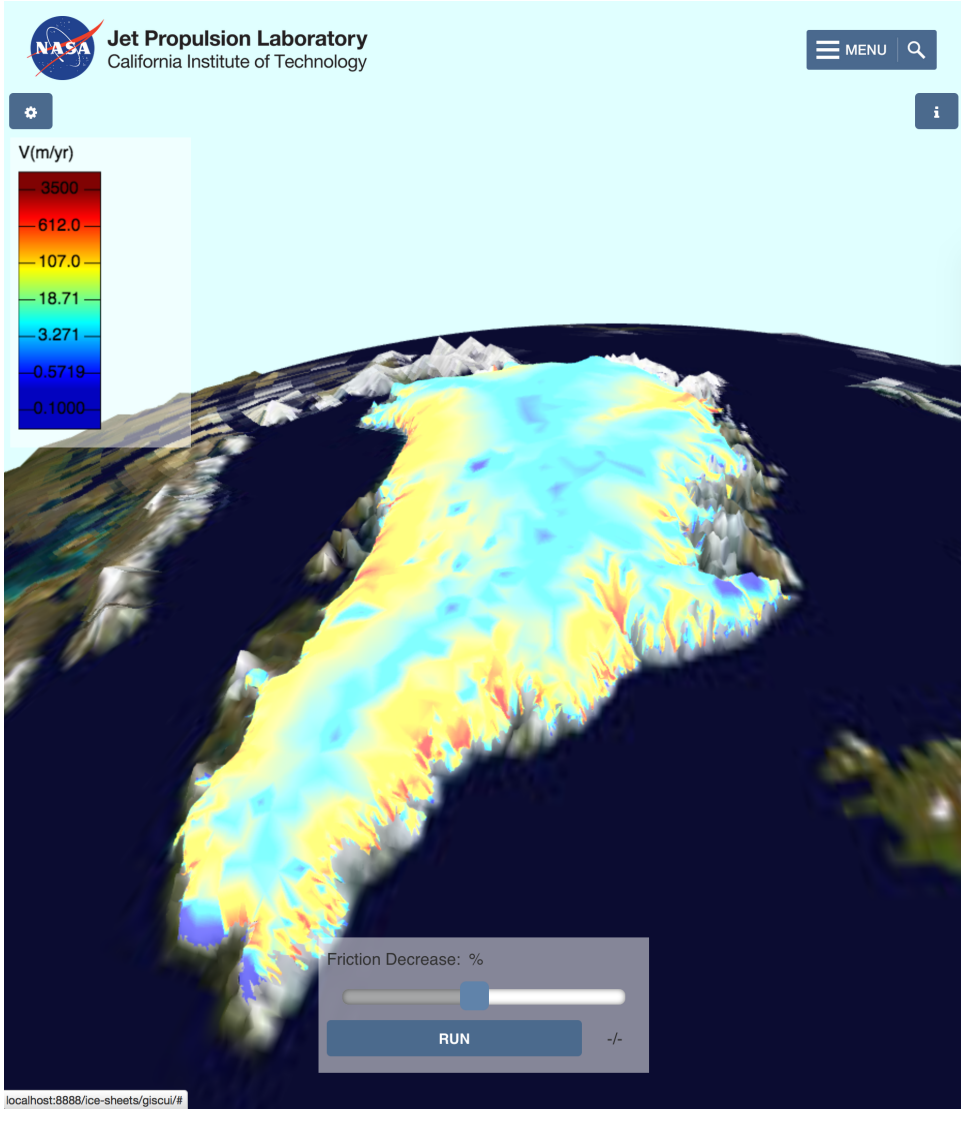

Figure 4. Greenland ISSM simulation on the Virtual Earth System Laboratory (ESL) (http://issm.jpl.nasa.gov/earthsystemlaboratorynew). This particular simulation allows for the introduction of user-driven friction anomalies (using a slider ranging from 5 to 100%) on the steady-state stress-balance velocities for the entire Greenland Ice Sheet. The computations (upon clicking of the RUN button) are carried out on the ISSM computational server (where the model inputs are uploaded, and from which the results are downloaded locally to the client' Web browser). The steady-state velocities are displayed for each value of the friction coefficient that the user chooses. The interactive rendering of the velocity field is done in 3D using the ISSM WebGl rendering engine. The results are overlaid on a semi-transparent topographical rendering of ETOPO5 data (see reference: National Geophysical Data Center (1988) for credits) and a background geotiff image from the Blue Marble: Land Surface, Shallow Water and Shaded Topography project (see reference: NASA Goddard Space Flight Center, Reto Stockli for credits).

References

Adhikari, S. and Marshall, S. J.: Improvements to shear-deformational models of glacier dynamics through a longitudinal stress factor, J. Glaciol., 57, 1003–1016, 2011.

Adhikari, S., Ivins, E. R., and Larour, E.: ISSM-SESAW v1. 0: mesh-based computation of gravitationally consistent sea-level and geodetic signatures caused by cryosphere and climate driven mass change, Geoscientific Model Development, 9, 1087–1109, doi:10.5194/gmd-9-1087-2016, 2016.

Bindschadler, R., Nowicki, S., Abe-Ouchi, A., Aschwanden, A., Choi, H., Fastook, J., Granzow, G., Greve, R., Gutowski, G., Herzfeld, U., Jackson, C., Johnson, J., Khroulev, C., Levermann, A., Lipscomb, W., Martin, M., Morlighem, M., Parizek, B., Pollard, D., Price, S., Ren, D., Saito, F.and Sato, T., Seddik, H., Seroussi, H., Takahashi, K., Walker, R., and Wang, W.: Ice-Sheet Model Sensitivities to Environmental Forcing and Their Use in Projecting Future Sea-Level (The SeaRISE Project), J. Glaciol., 59, 195–224, doi:10.3189/2013JoG12J125, 2013.

ECMA International: ECMAScript 2016 Language Specification, http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf, 2016.

Eyring, V., Bony, S., Meehl, G. A., Senior, C. A., Stevens, B., Stouffer, R. J., and Taylor, K. E.: Overview of the Coupled Model Intercomparison Project Phase 6 (CMIP6) experimental design and organization, Geoscientific Model Development, 9, 1937–1958, doi:10.5194/gmd-9-1937-2016, http://www.geosci-model-dev.net/9/1937/2016/, 2016.

Gropp, W., Lusk, E., Doss, N., and Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard, Parallel Computing, 22, 789–828, 1996.

Gropp, W. D. and Lusk, E.: User's Guide for `mpich`, a Portable Implementation of MPI, Mathematics and Computer Science Division, Argonne National Laboratory, aNL-96/6, 1996.

Inc, A.: Amazon Elastic Compute Cloud (Amazon EC2), Amazon Inc., http://aws.amazon.com/ec2/#pricing, http://aws.amazon.com/ec2/#pricing, 2008.

Jenkins: Jenkins, open source automation server, https://jenkins.io, 2016.

Larour, E. and Schlegel, N.: On ISSM and leveraging the Cloud towards faster quantification of the uncertainty in ice-sheet mass balance projections., Computers and GeoSciences, submitted, 2016.

Larour, E., Seroussi, H., Morlighem, M., and Rignot, E.: Continental scale, high order, high spatial resolution, ice sheet modeling using the Ice Sheet System Model (ISSM), J. Geophys. Res., 117, 1–20, doi:10.1029/2011JF002140, 2012.

Larour, E., Cheng, D., Perez, G., Quinn, J., Morlighem, M., Duong, B., Nguyen, L., Petrie, K., Harounian, S., Halkides, D., and Hayes, W.: Virtual Earth System Laboratory Website, http://vesl.jpl.nasa.gov, 2016.

MacAyeal, D.: Binge/Purge oscillations of the Laurentide ice-sheet as a cause of the North-Atlantic's Heinrich events, Paleoceanography, 8, 775–784, 1993.

Market, O.: Fastcgi: A high-performance web server interface, Technical white paper, 1996.

Molod, A., Takacs, L., Suarez, M., and Bacmeister, J.: Development of the GEOS-5 atmospheric general circulation model: evolution from MERRA to MERRA2, Geoscientific Model Development, 8, 1339–1356, doi:10.5194/gmd-8-1339-2015, http://www.geosci-model-dev.net/8/1339/2015/, 2015.

NASA Goddard Space Flight Center, Reto Stockli: NASA Goddard Space Flight Center Image by Reto Stöckli (land surface, shallow water, clouds). Enhancements by Robert Simmon (ocean color, compositing, 3D

globes, animation). Data and technical support: MODIS Land Group; MODIS Science Data Support Team; MODIS Atmosphere Group; MODIS Ocean Group Additional data: USGS EROS Data Center (topography); USGS Terrestrial Remote Sensing Flagstaff Field Center (Antarctica); Defense Meteorological Satellite Program (city lights).

National Geophysical Data Center: Data Announcement 88-MGG-02, Digital relief of the Surface of the Earth. NOAA, National Geophysical Data Center, Boulder, Colorado, 1988, 1988.

Nowicki, S., Payne, A., Larour, E., Seroussi, H., Goelzer, H., Lipscomb, W., Gregory, J., Abe-Ouchi, A., and Shepherd, A.: Ice Sheet Model Intercomparison Project (ISMIP6) contribution to CMIP6 , Geosci. Model Dev., 9, 4521–4545, doi:10.5194/gmd-9-4521-2016, 2016.

OpenMP Architecture Review Board: OpenMP Application Program Interface Version 4.5, http://www.openmp.org/mp-documents/openmp-4.5.pdf, 2015.

Pattyn, F.: GRANTISM: An ExcelTM model for Greenland and Antarctica ice-sheet response to climate changes, Comp. Geosci., 32, 316–325, 2005.

Taylor, K., Stouffer, R., and Meehl, G.: An Overview of CMIP5 and the experiment design, Bull. Amer. Math. Soc., 93, 485–498, doi:10.1175/BAMS-D-11-00094.1, 2012.

Taylor, K. E., Stouffer, R. J., and Meehl, G. A.: A Summary of the CMIP5 Experiment Design, 2009.

Vaughan, G. V., Elliston, B., Tromey, T., and Taylor, I. L.: GNU Autoconf, Automake, and Libtool, Pearson Education, http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1578701902, 2000.

World Wide Web Consortium: HTML 4.0 Specification – W3C Recommendation – Conformance: requirements and recommendations, https://www.w3.org/TR/REC-html40-971218/conform.html, 1997.

Zakai, A.: Emscripten: An LLVM-to-JavaScript Compiler, in: Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion, OOPSLA '11, pp. 301–312, ACM, New York, NY, USA, doi:10.1145/2048147.2048224, http://doi.acm.org/10.1145/2048147.2048224, 2011.

Manuscript prepared for Geosci. Model Dev.
with version 2015/04/24 7.83 Copernicus papers of the LaTeX class copernicus.cls.
Date: 27 January 2017

A JavaScript API for the Ice Sheet System Model (ISSM) 4.11: towards ~~on~~ an online interactive model for the Cryosphere Community

Eric Larour[1], Daniel Cheng[2], Gilberto Perez[2], Justin Quinn[2], Mathieu Morlighem[3], Bao Duong[4], Lan Nguyen[5], Kit Petrie[1], Silva Harounian[6], Daria Halkides[7], and Wayne Hayes[2]

[1]Jet Propulsion Laboratory - California Institute of technology, 4800 Oak Grove Drive MS 300-323, Pasadena, CA 91109-8099, USA
[2]University of California, Irvine, Department of Information and Computer Sciences, Donald Bren Hall, Irvine, CA 92697-3100, USA
[3]University of California, Irvine, Department of Earth System Science, Croul Hall, Ivine, CA 92697-3100, USA
[4]Monoprice, Inc.11701 6th Street Rancho Cucamonga, CA 91730, USA.
[5]Hart, Inc., 1515 E Orangewoord ave Anaheim, CA 92805 USA
[6]Digitized Schematic Solutions, Address: 40 W. Cochran st. Suite: 212 Simi Valley CA 93065, California, USA
[7] Earth and Space Research, 2101 Fourth Ave., Suite 1310, Seattle, WA 98121, USA.

Correspondence to: Eric Larour (eric.larour@jpl.nasa.gov)

Abstract.

Earth System Models (ESMs) are becoming increasingly complex, requiring extensive knowledge and experience to deploy and use in an efficient manner. They run on high-performance architectures that are significantly different from the everyday environments that scientists use to pre and post-process results (i.e. MATLAB, Python). This results in models that are hard to use for non specialists, and that are increasingly specific in their application. It also makes them relatively inaccessible to the wider science community, not to mention to the general public. Here, we present a new software/model paradigm that attempts to bridge the gap between the science community and the complexity of ESMs, by developing a new JavaScript Application Program Interface (API) for the Ice Sheet System Model (ISSM). The aforementioned API allows Cryosphere Scientists to run ISSM on the client-side of a webpage, within the JavaScript environment. When combined with a Web server running ISSM (using a Python API), it enables the serving of ISSM computations in an easy and straightforward way. The deep integration and similarities between all the APIs in ISSM (MATLAB, Python, and now JavaScript) significantly shortens and simplifies the turnaround of state-of-the-art science runs and their use by the larger community. We demonstrate our approach via a new Virtual Earth System Laboratory (VESL) Web site.

1 Introduction

Earth System Models (ESMs) across the Earth science community have become increasingly sophisticated, enabling more accurate simulations and projections of the Earth's climate as well as the state of the atmosphere, ocean, land, ice, and biosphere. As demonstrated by the Coupled Model Intercomparison Project 5 (CMIP-5, Taylor et al., 2009, 2012) and its new iteration (CMIP-6, Eyring et al., 2016) of the World Climate Research Programme (WCRP), the multiplicity of ESMs, and the complexity of the physics they capture, is significant. The description of the outputs for CMIP-5 runs is 133 pages long by itself, showing the complexity and comprehensive nature of the processes modeled in the ESMs that participated in the project. Any one of these models is massive both in terms of the number of lines of code, but also in terms of structure and modularity (or lack thereof). GEOS-5 for example (Molod et al., 2015), one of the Atmosphere and Ocean General Circulation Models (AOGCMs) that participated in CMIP-5, is made of 600,000 lines of Fortran code, comprising 88 physical modules (as of Jan 2016). This is fairly representative of the complexity of ESMs nowadays, and of the multiplicity of physical processes necessary to realistically model the evolution of the whole Earth System.

The above described complexity results in serious issues regarding the way simulations are run. For example, what we generally define as pre-processing and post-processing phases are increasingly different from the computational phase itself. The computational core is usually written in C or Fortran, which easily supports parallelism and High Performance Computing (HPC). However, in the pre-processing phase, where datasets are processed into a binary file used by the computational core, or in the post-processing phase, where simulation results are visualized, scientific environments such as MATLAB or Python are increasingly relied upon. ~~Indeed, presently, these environments are almost entirely ubiquitous within the science community.~~ This results in additional complexity to manage different environments: scientists are well-acquainted with the difficulties of porting their software to HPC instances, while struggling to process the data inputs and results on local workstations where data upload/download can be a limiting factor, hard drive memory requirements substantial, and problems due to the use of different APIs significant (MATLAB, Python, and IDL, among others).

Another complexity originating from the wide variety of physical processes represented in ESMs is the difficulty in initializing a computational run. For example, in the Ice Sheet System Model (ISSM, Larour et al., 2012), one of the land ice components of GEOS-5, developed at the National Aeronautics and Space Administration (NASA) Jet Propulsion Laboratory (JPL), in collaboration with University of California, Irvine (UCI), the initialization setup for the Greenland Ice Sheet (GIS) transient simulations from 1850 to present day amounts to 3,000 lines of MATLAB code. This comprises model setup, data interpolation onto an ISSM compatible mesh, solution parameterizations, and initialization strategies, among other things. This simulation, part of the Ice Sheet Modeling Intercomparison Project 6 (ISMIP-6 Nowicki et al., 2016) that accounts for ice sheets in CMIP-6,

is a fairly representative example of some of the most advanced simulations that can be run with an Ice Sheet Model (ISM). Such simulations cannot easily be systematized and need to be tailored specifically for each ice sheet they are applied to. ~~Our JavaScript API, however, allows a scientist to handle sophisticated computations without having to become an expert in each and every one of the software modules required by the physics of a given simulation.~~

One of the approaches that could mitigate some of the issues discussed above involves the development of computational frameworks capable of serving ESM simulations. This type of solution involves running simulations that already include pre and post-processing phases (i.e.where the model setup has already been carried out or is carried out by the server itself by uploading key datasets) and in which the user is allowed to control only a few, key parameters. Similarly, once the computation is carried out on the server-side, the results are post-processed automatically, and only significant results are provided to the user. This type of approach has already been explored, for example, in areas relating to serving of large datasets, such as the NASA Earth Observing System Data and Information System (EOSDIS) EarthData server, which provides a portal with integrated processing capabilities for large scale datasets collected by NASA missions. However, fewer examples of this kind of approach are available that serve simulation results, and to our knowledge, no comprehensive ESM, nor module thereof, has ever been integrated into a server solution capable of delivering ESM computations on the fly. The reason for this is simple: the complexity of the physics involved is significant, reconciling pre/post processing phases and simulation cores is inherently difficult, and basing a simulation framework on server technologies represents a significant software development challenge.

Specifically, the bottlenecks that preclude deeper integration of ESMs within server infrastructures include: ~~1)~~

1. Bridging the gap between ESM formulations of the physical cores and Web technologies such as HyperText Markup Language (HTML, World Wide Web Consortium, 1997) and JavaScript (ECMA International, 2016), which are not ~~languages that are~~ scientifically oriented languages and are thus not inherently used by Earth scientists~~, for reasons that are obvious~~. Because ESMs are not natively integrated into Web technologies, it renders the link between server infrastructures and simulation engines difficult ~~; 2)~~

2. The significant turnaround between generation and serving of simulations. This lag is due to the fact that these two processes are inherently different in the way they are designed and, moreover, are usually considered to be completely separate phases of what should, essentially, be the same process~~; 3)~~ .

3. The distributed nature of Web simulations. Every step of an ESM run can be considered a separate, logical component. For example, post-processing of a simulation may be done on a different machine than the one that initially generated it~~; and 4)~~ .

4. The lack of existing integrated frameworks wherein simulations, pre and post-processing, and the serving of the data and/or simulation results all occur within the same architecture.

Here, we present a new approach applied to the ISSM framework, a land ice model of significant size and complexity, to serve ~~simulation results~~ simulations relating to the evolution of polar ice sheets. Here, by serving, we imply providing a way to run simulations interactively within a Web environment, without any of the results ever being cached. Our solution is based on a new JavaScript API for the ISSM framework itself, allowing it to be fully integrated within ~~a Web environment, namely~~ an HTML webpage (described in Section 2) and to run local to the webpage. For models of larger size, we also show how we leverage the existing ISSM Python API to run a web server (based on Apache and the FastCGI module) that can run faster parallel computations, and to which the webpage client can upload model inputs and download computation along with pre and post-processing results directly (Section 3). ~~By leveraging its existing Python API within a FastCGI module of an Apache HTTP server , we show (in Section 3) how ISSM can be used to provide simulations and to pre and post-process results directly .~~ This new approach allows for a quick turnaround between ~~model setup and simulation, in particular shortening the time required to carry out science runs and to serve the results to the~~ running simulations and porting such simulations to a webpage interface for access to the wider science community (Section 4). We execute this approach (Section 5) within the newly-designed Virtual Earth System Laboratory (VESL), demonstrating how we can provide access to cryosphere-related simulations to the science community, and to the wider public in general, thereby easily providing access to the wide array of modular physics embedded in ISSM. We conclude with a discussion of the potential of this new approach to both facilitate a wider use of ESMs by scientists of varied disciplines and to shorten the gap between science simulations and public outreach.

2 ISSM JavaScript API

Most ISMs are written in Fortran, C, or C++, for reasons related to computational efficiency and to the ease of integration within HPC environments using parallel libraries, such as Message Passing Interface (MPI) via OpenMP (Gropp et al., 1996; Gropp and Lusk, 1996; OpenMP Architecture Review Board, 2015). However, many simpler models exist that rely on different APIs, such as the MATLAB code described in MacAyeal (1993) or the Excel-based Greenland and Antarctica Ice Sheet Model designed for educational purposes (GRANTISM, Pattyn, 2005). These models have in common the desire to rely on a simple code base, and to reduce/optimize the set of physics captured in the code, in order to make it more accessible. Our approach here, however, is to facilitate accessibility without sacrificing the complexity and full-set of features of ISSM by implementing a brand new API using the JavaScript language. The goal is to be able to integrate ISSM within Web-based solutions, relying on JavaScript as a language that enables control of the behavior of an

HTML webpage. In addition, by making the JavaScript API similar in all possible aspects to the existing MATLAB and Python ISSM APIs, model runs and simulations can be transferred easily to the Web, furthering our objective of disseminating ISSM to the larger scientific community and, possibly, the general public ~~.~~ through Web interfaces. It is to be noted that the new API being of equivalent complexity (to capture the full range of physics) to the MATLAB or Python APIs, users that want to use this API should be fully knowledgeable with using ISSM in MATLAB or Python already. This means that the new API does not make use of ISSM easier in terms of learning curve, but makes it more flexible in terms of being deployed to the Web.

The basis for representing a model in ISSM is a series of classes (mesh, mask, geometry, settings, toolkits, etc.) that are carried into a global `model` class. The first task was, therefore, to translate all ISSM classes from MATLAB and Python into JavaScript. Fig. 1 shows an example of such a translation for the `mesh2d` class (used to represent a 2D mesh triangulation comprising a list of vertex coordinates `x`,`y` of size `numberofvertices` with corresponding `lat`,`long` coordinates, a list of triangle indices called `elements` (of size `numberofelements`), and a projection code using an EPSG Geodetic Parameter Dataset). The constructors are very similar, and there is a one-to-one correspondence between the mesh2d methods in both APIs. The example of the `marshall` routine (which collects all the mesh info onto a binary buffer that will be sent to the ISSM C++ core) shows the similarity between both codes, with differences in the syntax reduced to a bare minimum. This equivalence is essential in preserving all of the physics captured in each class of ISSM, and could only be achieved because MATLAB, Python, and JavaScript are similar in their syntax and grammar.

~~The~~ In a standard modeling analysis, scientists will develop their models and run within the MATLAB (or Python) environment. Usually, outreach of the results will be done separately, in a different web based environment, leading to inefficiencies and potential loss of information/accuracy between the science analysis and the outreach itself. To remedy this issue, it is very convenient to provide an efficient way to transfer a model directly from MATLAB to the JavaScript environment, where it will be loaded easily using a standard 'include' statement. This is implemented through the `savemodel` routine ~~in the MATLAB~~ for each subclass of the `model` class. As shown in Fig. 1 for the `mesh2d` class implementation~~is unique, as it~~, the `savemodel` routine allows users to write the MATLAB model to a JavaScript file. This allows users to run simulations in MATLAB using ISSM, and, once the simulations are over, to save the MATLAB defined model into a JavaScript equivalent file. This routine, which closely matches the constructor, is the key to shortening the transition time between the setup of an ISSM simulation and its transition ~~to~~ into a webpage environment. The fact that all of the information of a given class is identical in both APIs demonstrates the comprehensiveness of the new JavaScript implementation of ISSM, and that it achieves its goal of replicating ISSM within a webpage environment.

In ~~addition to the classes representation in JavaScript, ISSM relies on a~~ a standard model run, MATLAB classes (or Python) are used to setup the model, but the computations themselves are carried out in C++~~core for computations~~. This C++ code is present at several levels: 1) For each pre and post-processing module (or, wrapper) that requires significant computational power, such as interpolation routines that transfer information between gridded dataset and unstructured Finite Element Modeling (FEM) meshes typical of ISSM; and 2) For each of the computations pertaining to ice flow itself (the physical engine in ISSM), which we refer to as the ISSM core. For ~~both sets of code, a solution relying on the~~ pre and post-processing modules, the computations are assumed local to the workstation. For the ISSM core itself, parallelization is inherent (using the MPI libraries), and this core usually runs on a parallel cluster.

When we look at this configuration and try and transfer this paradigm to a webpage environment, we are however faced with two issues: 1) C++ code cannot be run native to a webpage easily and 2) parallelism is not yet implemented in browsers, and would anyway result in heavy taxation of CPU ressources (on local workstations/laptops/tablets), which is not practical. We therefore approached this issue in two ways: 1) we translated the entire C++ code (both modules/wrappers and the ISSM core itself) into JavaScript for model runs that are small enough to be run locally; and 2) for models that are too large to run local, we implemented a way of uploading (using the JavaScript classes) a model to a web server on the Amazon EC2 cloud, where computations are carried out and returned to the JavaScript client once completed. The latter approach is described in the next section. We here further describe the translation of the C++ modules and ISSM core into JavaScript code. This translation was carried out using the Emscripten compiler (Zakai, 2011) ~~was deployed~~. This compiler enables translation of C++ code directly into JavaScript, with computational efficiencies that are within an order of magnitude of the translated C++ code. Listing 1 shows how Emscripten was integrated within the existing Makefile structure of ISSM. All the pre and post-processing wrappers (TriMesh, NodeConnectivity, ContourToMesh, ElementConnectivity, InterpFromMeshToMesh2d, IssmConfig, EnumToString, and StringToEnum) as well as the ISSM core itself (issm) are compiled into JavaScript executables using the C++ files and a set of Emscripten related flags (described in the IssmModule_CXXFLAGS variable). This Makefile is similar to its MATLAB and Python counterparts, with the exception of the issm core, which is compiled as a JavaScript module instead of a C++ executable. This Makefile is integrated within Autotools (Vaughan et al., 2000), enabling for quick activation of the compilation using a simple "`--with-javascript`" option during the configuration phase of the ISSM software.

The JavaScript modules and ISSM core are continuously tested against regression tests, similar to the MATLAB and Python APIs (Larour et al., 2012). The integration framework for the tests relies on Jenkins, an open-source automation server (Jenkins, 2016), which provides continuous integration and delivery of validated ISSM code. The ISSM Jenkins webpage is available at https://ross.ics.uci.edu:8080/, where the entire validation suite is in the process of being transferred to

JavaScript. This ensures that continuous development impacts all of the APIs in ISSM in a similar fashion without imparting delays to the JavaScript API (due to the fact that it would be used by a smaller base of ISSM users).

3 HTTP/Python Server

Using the JavaScript API, it is possible to run a full-fledged simulation using any of the physical modules described in Larour et al. (2012). However, to our knowledge, Emscripten does not yet allow computations in parallel within a browser. This limits the range of model sizes and mesh resolution to a level that compromises large-scale simulations. In these cases, our approach was to rely on the cloud computing capabilities of ISSM, as described in Larour and Schlegel (2016), and to host a Web server that would deliver ISSM computations to any client running the ISSM JavaScript API. This server relies on the Python API of ISSM to carry out computations ranging from tens to hundreds of thousands of degrees of freedom, allowing continental-scale simulations. The server is fully-elastic and scalable, and relies on the Amazon EC2 infrastructure (Inc, 2008), and can spin-up Compute Optimized CC4.8x large instances (up to 64 threads of computational power) on demand, making it a robust solution for serving computations. Refer to Larour and Schlegel (2016) for more details on this part of the architecture.

In terms of server configuration itself, our approach was to rely on the Python API of ISSM to leverage the FastCGI Web interface, described in Market (1996), on an Apache server. This allows requests coming into the Apache server from the client-side to be routed directly to a Python script. The Web client, running ISSM embedded inside JavaScript, can therefore upload a marshalled binary input file (created by the call to the marshall routine of each model class, as described in Fig. 1) to the EC2 instance Apache server, which then routes it to the Python script that launches the parallel job.

Fig. 2 describes this process schematically, and compares it to what happens in more classic simulations relying on MATLAB and an HPC infrastructure, such as a cluster. The fundamental differences between the traditional simulation paradigm and our new solution are: 1) The client architecture, which runs either MATLAB or an HTML webpage with JavaScript; 2) The upload/download of binary input files, which is done either through an SSH copy call or an XMLHttpRequest, respective to the aforementioned client architectures; and 3) The launching of a given computation, which is handled via a queuing system on the head node or a FastCGI-relayed Python call on an EC2 instance, again, respective to the client architecture. In terms of parallel computations, ISSM executables are run using an MPI call in both cases. The strong similarity between both architectures was purposefully designed so as to limit the amount of repeated code, and to ensure the robustness of the computations themselves, which are transparent to the API they rely upon.

4 All-In-One Design/Simulations

Listing 2 shows a typical model setup for a simulation in ISSM relying on the MATLAB API. The steps include loading a model (or generating one using a mesher), modifying a certain input parameter, ~~such as surface mass balance (SMB),~~ setting up a cluster class (pointing to the parallel cluster) and calling the solve routine. Once the results are carried out/downloaded, plotmodel is run to visualize them.

An additional step can be carried out once a given MATLAB ISSM model has been built, wherein the model is saved into a JavaScript file (md.js) in some webpage directory. This model can then be used (as shown in Listing. 3) to run the exact same setup and simulation as is done with MATLAB, but on the client's machine. The HTML code for this simulation is typical of a webpage, and includes: 1) Standard HTML markup (i.e. W3C-compliant html, head, and body objects); 2) Include statements for the ISSM binaries created by Emscripten, the model itself (md.js), and a sort of front-end controller (engine.js, which controls the display of and interaction with the simulation on the webpage); and 3) HTML elements such as a canvas where the results will be plotted (similar to the figure statement in MATLAB), a second canvas for the color bar, and a button element to launch the simulation. The listing for engine.js shows how similar the MATLAB and JavaScript setup are. Upon loading, if the RUN button is clicked, the value of a slider (~~SMB value~~the model input of interest) is retrieved and then SolveGlacier called. The SolveGlacier() routine modifies the ~~SMB~~ input parameter, sets up the cluster class (pointing to the EC2 server), and calls the solve routine. After computations are carried out and downloaded, a callback function PlotGlacier is triggered, which plots the model results onto the aforementioned HTML canvas elements. If users do not want to rely on this particular routine for plotting, they can instead provide their own callback routine to plot using their own rendering engine.

Fig. 3 shows an example of such a webpage hosting a simulation ~~on the impact of anomalies in SMB on transient ice flow(including Mass Transport and Stress Balance)on the Columbia Glacier in Alaska. This~~ for the Columbia Glacier, Alaska. In this particular example, the model input that is modified is the surface mass balance (SMB). This parameter measures the amount of precipitation (in snow or water) at the surface of the ice, minus runoff of water from melting and evaporation. This parameter is essential in controling the input of mass to the glacier. Once this input is modified, we can measure its impact on the response of the glacier (the ice flow) through time. This response is a complex interplay between mass transport processes and the stress equilibrium of the ice. The result is a new flow regime (speed), which ISSM can compute and which can be visualized through a time evolution of the speed at the surface of the ice.

Here, the webpage is part of VESL, where the JavaScript API of ISSM was leveraged along with the HTTP/Python Server architecture described previously to showcase the capabilities of ISSM to serve computations on the fly and to visualize them instantly (Larour et al., 2016). The simulations within VESL are all simulations that were carried out using ISSM for scientific publications.

By adding a savemodeljs step at the end of the MATLAB simulation workflow, ~~and by replicating some of that same workflow~~ we were able to transfer the model used for the simulations from the MATLAB environment onto the webpage. Once that was done, we replicated a workflow similar to the MATLAB workflow in the engine.js code. With this approach, it is possible to deploy a simulation like the one described above on a Web platform with significantly shortened turnaround and using the exact same capabilities as the initial MATLAB solution itself. This breakthrough is only possible because of the duplication of the entire architecture: again, by making JavaScript code that is logically equivalent to our MATLAB or Python constructs and by mapping the whole workflow described in Fig. 2 from MATLAB/HPC infrastructures to HTML/JavaScript/EC2. Our methodology paves the way to leveraging Web technologies and cloud computing to host large-scale simulations of modeling engines such as ISSM, all without loss of the physical representation of processes nor scalability.

5 Examples

Fig. 3 and Fig. 4 show examples of simulations that rely on the ISSM JavaScript API, and that are hosted on the VESL Web site (Larour et al., 2016). VESL's purpose is to twofold: to showcase simulations that demonstrate ISSM capabilities, and to demonstrate the capabilities of our new Web-based modeling solution to the wider scientific community and general public. Several simulations are hosted, leveraging the large set of capabilities in ISSM.

The first simulations pertain to the simulation of glacier flow, mainly from work on Haig Glacier (Adhikari and Marshall, 2011) and Columbia Glacier (Gardner, Fahnestock, Larour, pers. comm.). Fig. 3 shows the Columbia Glacier webpage~~is~~, where SMB ~~anomalies (to the background trend)~~ variations can be specified, with ISSM then computing the resulting ~~modification to the transient flow~~ impact on the glacier evolution over a period of 10 years. This simulation includes ~~mass transport, stress balance, a basal friction parameter, which was inverted using surface velocities (Fahnestock, Gardner and Larour, pers. comm.) following Morlighem et al. (2013) , and thermal steady-state.~~ all the physical processes that control the evolution of a glacier, and is fully representative of the complex physics required in the analysis.

The second set of simulations pertain to ice sheet modeling in Antarctica and Greenland. Fig. 4 shows the webpage corresponding to the friction SeaRISE (Bindschadler et al., 2013) experiment over the entire Greenland ice sheet. In this simulation, ~~which is also calculated using a basal friction inversion from surface velocities (Rignot, 2008) , thermal steady-state, and stress-balance (no mass transport, nor transient),~~ the user can decrease the friction ~~under the ice~~ at the ice/bedrock interface under the ice sheet and compute the resulting changes in ~~surface velocities~~steady-state velocity at the surface. The model is fairly high resolution (12,000 elements), which allows for ~~physically-relevant computations~~computationas that are physically representative.

The third set of simulations pertains to Sea-Level Rise (SLR) modeling, relying on the ISSM-SESAW module (Adhikari et al., 2016) to compute gravitationally consistent sea-level and geodetic signatures caused by cryosphere and climate-driven mass change. Presently, two sets of simulations demonstrate: 1) Eustatic SLR and its impact on coastline migration in the USA; and 2) SLR from eustatic, gravity, and elastic deformation on a global scale, wherein users can turn off specific sets of SLR physics to understand the impact of gravitation on redistribution of SLR around the world and the impact of local elastic deformation of the Earth lithosphere.

Finally, a fourth set of simulations pertains to Solid Earth deformation, using the ISSM-GIA (Adhikari and Marshall, 2011) module that captures Glacial Isostatic Adjustment (GIA) from ice-sheet loading. It should be noted that this section is a work in progress.

One potential future section may feature recent work by the ISSM team involving the application of ISSM to other planets (namely, Mars' ice caps). Given the relatively quick turnaround between ISSM simulations and their porting to the Web using the ISSM JavaScript API, our hope is that VESL will become a forum for cryosphere scientists to discuss ice sheet related science. In addition, by enabling simplified interfaces on existing simulations that resulted in scientific publications, we believe the general public might gain increased interest in this type of approach to better understand the complexities of science for the Earth system as a whole.

6 Conclusions

We developed a fully-functional JavaScript API for the Ice Sheet System Model (ISSM), which allows cryosphere scientists to carry out ice flow simulations within a Web environment. This API gives access to the entire spectrum of physical processes captured by ISSM without compromising its complexity and richness. For simulations requiring parallel computing, the JavaScript API can be leveraged against a computational server hosted on a cloud instance (such as Amazon EC2) to deliver high-performance, large-scale, and high-fidelity simulations back to the Web client. This new set of capabilities enables hosting of high-end simulations on the NASA/JPL ESL, effectively solving a fundamental challenge of ESMs: delivering accessible, high-performance simulations in a timely manner is historically and inherently difficult. We believe that our approach paves the way for the efficient deployment of feature-rich ESM's, a quick turnaround between scientific work and corresponding publications, and outreach not only the science community but also to the general public.

7 Code Availability

The ISSM code and its JS components are available at http://issm.jpl.nasa.gov. The instructions for the compilation of ISSM in JS mode is presented in the supplement attached to this manuscript.

Acknowledgements. This work was performed at the Jet Propulsion Laboratory (JPL), California Institute of Technology, and the Department of Earth System Science at the University of California, Irvine (UCI) under a contract with the National Aeronautics and Space Administration (NASA) and funded by the Cryospheric Sciences Program. Resources supporting the numerical simulations were provided by the NASA High-End Computing (HEC) Program through the NASA Advanced Supercomputing (NAS) Division at Ames Research Center, and by Cryospheric Program Management for the Amazon EC2 instances hosting the Virtual Earth System Laboratory. We would like to thank Dr. Alex Gardner from JPL and Dr. Mark Fahnestock from the University of Alaska, Fairbanks for the datasets used in the Columbia Glacier model setup of the Virtual Earth System Laboratory. We would also like to thank Dr. Daisy F. Sang for the supervision of students from CalPoly Pomona who participated in this project over the past 5 years.

Listing 1. Makefile for Javascript Emscripten compilation of ISSM.

```
EXEEXT=.js

js_scripts = ${ISSM_DIR}/src/wrappers/TriMesh/TriMesh.js \
${ISSM_DIR}/src/wrappers/NodeConnectivity/NodeConnectivity.js\
${ISSM_DIR}/src/wrappers/ContourToMesh/ContourToMesh.js\
${ISSM_DIR}/src/wrappers/ElementConnectivity/ElementConnectivity.js\
${ISSM_DIR}/src/wrappers/InterpFromMeshToMesh2d/InterpFromMeshToMesh2d.js\
${ISSM_DIR}/src/wrappers/IssmConfig/IssmConfig.js\
${ISSM_DIR}/src/wrappers/EnumToString/EnumToString.js\
${ISSM_DIR}/src/wrappers/StringToEnum/StringToEnum.js\
${ISSM_DIR}/src/wrappers/Issm/issm.js

bin_SCRIPTS  = issm-prebin.js
bin_PROGRAMS = IssmModule

issm-prebin.js: ${js_scripts}
  cat ${js_scripts} > issm-prebin.js

IssmModule_SOURCES = ../TriMesh/TriMesh.cpp \
             ../NodeConnectivity/NodeConnectivity.cpp\
             ../ContourToMesh/ContourToMesh.cpp\
             ../ElementConnectivity/ElementConnectivity.cpp\
             ../InterpFromMeshToMesh2d/InterpFromMeshToMesh2d.cpp\
             ../IssmConfig/IssmConfig.cpp\
             ../EnumToString/EnumToString.cpp\
             ../StringToEnum/StringToEnum.cpp\
             ../Issm/issm.cpp

IssmModule_CXXFLAGS= -fPIC -D_DO_NOT_LOAD_GLOBALS_ --memory-init-file 0 \
```

```
$(AM_CXXFLAGS) $(CXXFLAGS) $(CXXOPTFLAGS) $(COPTFLAGS) \
-s EXPORTED_FUNCTIONS="['_TriMeshModule','_NodeConnectivityModule',\
'_ContourToMeshModule','_ElementConnectivityModule',\
'_InterpFromMeshToMesh2dModule','_IssmConfigModule','_EnumToStringModule'\
,'_StringToEnumModule','_IssmModule']"  -s DISABLE_EXCEPTION_CATCHING=0 \
-s ALLOW_MEMORY_GROWTH=1 -s INVOKE_RUN=0

IssmModule_LDADD = ${deps} $(TRIANGLELIB)  $(GSLLIB)
```

```
%MESH2D class definition
classdef mesh2d
    properties (SetAccess=public)
        x                           = NaN;
        y                           = NaN;
        elements                    = NaN;
        numberofelements            = 0;
        numberofvertices            = 0;
        numberofedges               = 0;
        lat                         = NaN;
        long                        = NaN;
        epsg                        = 0;
    end
    methods
+-- 18 lines: function self = mesh2d(varargin) % ----------------------------------------------------------------
+--  9 lines: function self = setdefaultparameters(self) % ------------------------------------------------------
+-- 19 lines: function md = checkconsistency(self,md,solution,analyses) % -------------------------------------
function marshall(self,md,fid) % {{{
WriteData(fid,'enum',DomainTypeEnum(),'data',StringToEnum(['Domain' domaintype(self)]),'format','Integer');
WriteData(fid,'enum',DomainDimensionEnum(),'data',dimension(self),'format','Integer');
WriteData(fid,'enum',MeshElementtypeEnum(),'data',StringToEnum(elementtype(self)),'format','Integer');
WriteData(fid,'object',self,'class','mesh','fieldname','x','format','DoubleMat','mattype',1);
WriteData(fid,'object',self,'class','mesh','fieldname','y','format','DoubleMat','mattype',1);
WriteData(fid,'enum',MeshZEnum(),'data',zeros(self.numberofvertices,1),'format','DoubleMat','mattype',1);
WriteData(fid,'object',self,'class','mesh','fieldname','elements','format','DoubleMat','mattype',2);
WriteData(fid,'object',self,'class','mesh','fieldname','numberofelements','format','Integer');
WriteData(fid,'object',self,'class','mesh','fieldname','numberofvertices','format','Integer');
end % }}}
+--  3 lines: function t = domaintype(self) % -------------------------------------------------------------------
+--  3 lines: function d = dimension(self) % --------------------------------------------------------------------
+--  3 lines: function s = elementtype(self) % ------------------------------------------------------------------
function savemodeljs(self,fid,modelname) % {{{
writejs1Darray(fid,[modelname '.mesh.x'],self.x);
writejs1Darray(fid,[modelname '.mesh.y'],self.y);
writejs2Darray(fid,[modelname '.mesh.elements'],self.elements);
writejsdouble(fid,[modelname '.mesh.numberofelements'],self.numberofelements);
writejsdouble(fid,[modelname '.mesh.numberofvertices'],self.numberofvertices);
writejsdouble(fid,[modelname '.mesh.numberofedges'],self.numberofedges);
writejs1Darray(fid,[modelname '.mesh.lat'],self.lat);
writejs1Darray(fid,[modelname '.mesh.long'],self.long);
writejsdouble(fid,[modelname '.mesh.epsg'],self.epsg);
end % }}}
end
end
```

```
//MESH2D class definition
function mesh2d () {
    //methods
+-- 10 lines: this.setdefaultparameters = function (){ ------------------------------------------------------
+--  3 lines: this.classname = function () { -------------------------------------------------------------------
+--  3 lines: this.domaintype=function (){ ---------------------------------------------------------------------
+--  3 lines: this.dimension = function () { -------------------------------------------------------------------
+--  3 lines: this.elementtype = function() {------------------------------------------------------------------
this.marshall=function(md,fid) { //{{{
WriteData(fid,'enum',DomainTypeEnum(),'data',StringToEnum('Domain' + this.domaintype()),'format','Integer');
WriteData(fid,'enum',DomainDimensionEnum(),'data',this.dimension(),'format','Integer');
WriteData(fid,'enum',MeshElementtypeEnum(),'data',StringToEnum(this.elementtype()),'format','Integer');
WriteData(fid,'object',this,'class','mesh','fieldname','x','format','DoubleMat','mattype',1);
WriteData(fid,'object',this,'class','mesh','fieldname','y','format','DoubleMat','mattype',1);
WriteData(fid,'enum',MeshZEnum(),'data',NewArrayFill(this.numberofvertices,0),'format','DoubleMat','mattype',1)
WriteData(fid,'object',this,'class','mesh','fieldname','elements','format','DoubleMat','mattype',2);
WriteData(fid,'object',this,'class','mesh','fieldname','numberofelements','format','Integer');
WriteData(fid,'object',this,'class','mesh','fieldname','numberofvertices','format','Integer');
}//}}}
+-- 12 lines: this.fix=function() { ---------------------------------------------------------------------------
    //properties
    // {{{
        this.x                          = NaN;
        this.y                          = NaN;
        this.elements                   = NaN;
        this.numberofelements           = 0;
        this.numberofvertices           = 0;
        this.numberofedges              = 0;

        this.lat                        = NaN;
        this.long                       = NaN;
        this.epsg                       = 0;

        this.setdefaultparameters();
        //}}}
}
~
```

Figure 1. Line by line comparison of the code behind the mesh2d class, within the MATLAB ISSM API (upper frame) and the JavaScript ISSM API (lower frame). Routines followed by a dashed line have been folded for ease of reading.

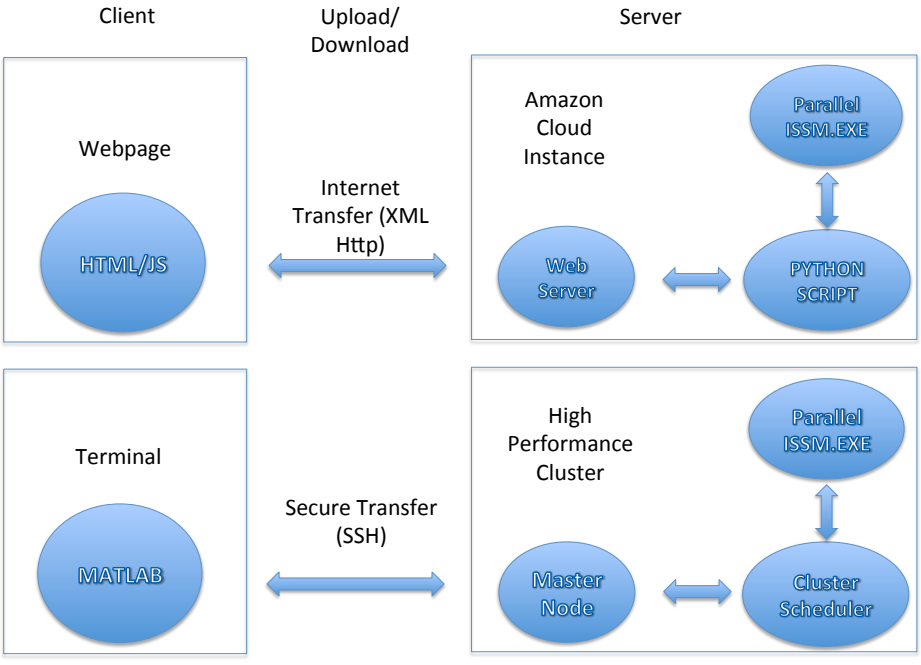

Figure 2. Similarities between a standard ISSM run from a terminal running MATLAB ~~/~~and connected to a high-performance cluster (HPC) and a web based ISSM run ~~and~~ from a Webpage runing JavaScript~~/~~, connected to a Web server runing on an Amazon EC2 ~~driven run~~Cloud instance. In the first case (lower frames) a MATLAB instance running on a local workstation terminal running ~~the ISSM API~~ marshalls an input binary file, which is then uploaded (using an ssh call) to a master node on a cluster. The binary file is then queued into the system (using a qsub command from a scheduler, for example). The parallel runs are then carried out using the ISSM executable and an MPI-compatible environment. In the second case, a browser client makes an XMLHttpRequest and uploads a Binary Large Object (the exact same binary file MATLAB would upload), which is received by an HTTP server (e.g. Apache) running on an Amazon EC2 compute-optimized instance. The HTTP server then uses a FastCGI module to interface to a Python wrapper, which automatically triggers a system call to the MPI environment running the ISSM executable. In both cases, an output binary file is created by the ISSM executable, which is then shipped back to the MATLAB instance or the client's Web browser.

;

Listing 2. MATLAB code for a typical simulation of the Virtual Earth System Laboratory (VESL).

```
% Load Model:
md=loadmodel('Models/md.mat');

% Solve:
```

```
md.smb.mass_balance= smb_initial;
for i=1:md.mesh.numberofvertices,
  md.smb.mass_balance(i) = md.smb.mass_balance(i)+ smbvalue;
end

md.cluster=generic('name','localhost','np',8);
md=solve(md,TransientSolutionEnum());

%Plot Model:
vel=md.results.TransientSolution(1).Vel;
plotmodel(md,'data',vel,'log',10,'figure',1,'colorbar','on',...
      'overlay','on','images','radar.png');

% Export to JS model:
md.savemodeljs('md',websiteroot);
```

Listing 3. Equivalent (see Listing. 2) Hmtl/Javascript code for a typical simulation within the Virtual Earth System Laboratory. Prototype webpage.

```
<html>
  <script type="text/javascript" src="./bin/issm-binaries.js"></script>
  <script type="text/javascript" src="./src/engine.js"></script>
  <script type="text/javascript" src="./js/md.js"></script>

  <body data-spy="scroll" data-target="nav" onload="engine();">

    <canvas id="columbia"></canvas>

    <div class="bordered margin-8 padding-8">
      <canvas id="columbia-colorbar" class="colorbar-v"> </canvas>
    </div>

    <div id="columbia-run" class="bordered margin-8 padding-8">
      <button type="button" class="interactive run-button"
        onclick="SolveGlacier()"> RUN </button>
    </div>
  </body>
</html>

function engine(){
  PlotGlacier();
  slider('value',0,'callback',function(value){smbvalue=value;},
```

```
  'name','columbia', 'min',-5,'max',+5,'message',['SMB anomaly:','m/a'],
  'step',.1, 'slidersdiv','columbia-sliders');
}

function SolveGlacier(){
  md.smb.mass_balance= smb_initial.slice(0);
  for (var i=0;i<md.mesh.numberofvertices;i++){
    md.smb.mass_balance[i] += smbvalue;
  }
  md.cluster=new generic('url',server + '/fastcgi/issm_solve.py','np',8);
  md=solve(md,TransientSolutionEnum(),'checkconsistency','no',
  'callback',PlotGlacier);
}

function PlotGlacier(){
  plotmodel(md,'data',md.results[0]['Vel'],'log',10,'canvasid#all','columbia',
    'colorbar','on', 'colorbarcanvasid','columbia-colorbar',
    'overlay','on','image','./images/radar.png');
}
```

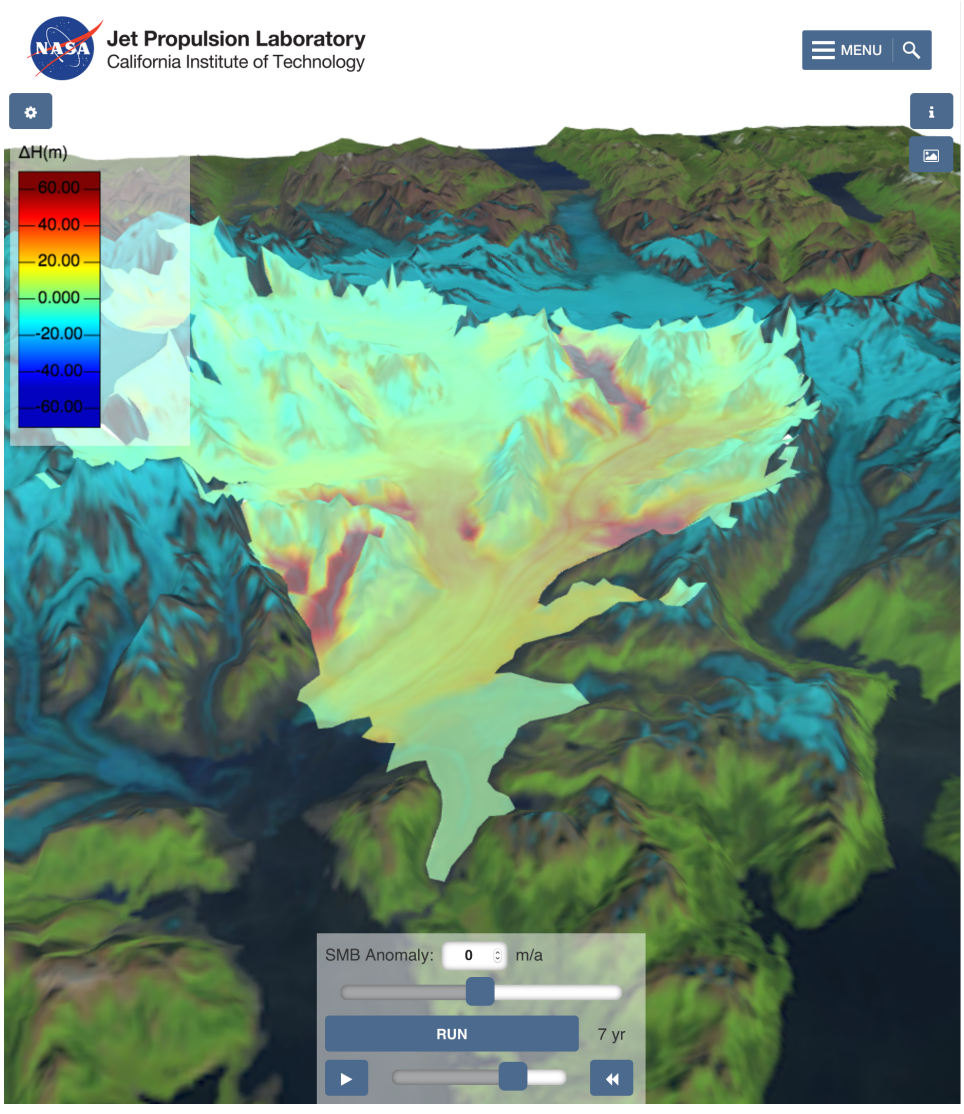

Figure 3. Columbia Glacier ISSM simulation on the Virtual Earth System Laboratory (http://issm.jpl.nasa.gov/earthsystemlaboratorynew). This particular simulation allows for the introduction of user-driven SMB anomalies (using a slider ranging from -5 to +5 m/a) on the transient ice flow of Columbia Glacier. The computations (upon clicking of the RUN button) are carried out on the ISSM computational server (where the model inputs are uploaded, and from which the results are downloaded locally to the client's Web browser). The transient results are displayed as a movie, which can be controlled via user interface (UI) controls. The interactive rendering of the velocity and thickness fields is done in 3D (or 2D, upon clicking of a toggle button) using the ISSM WebGL rendering engine. The results are overlaid on a semi-transparent topographical rendering of the SRTM DEM, and a background geotiff image from Gardner et al (pers. comm.). Model information can be displayed by clicking the info button, allowing for extensive information on the model setup and the datasets used to constrain the simulation.

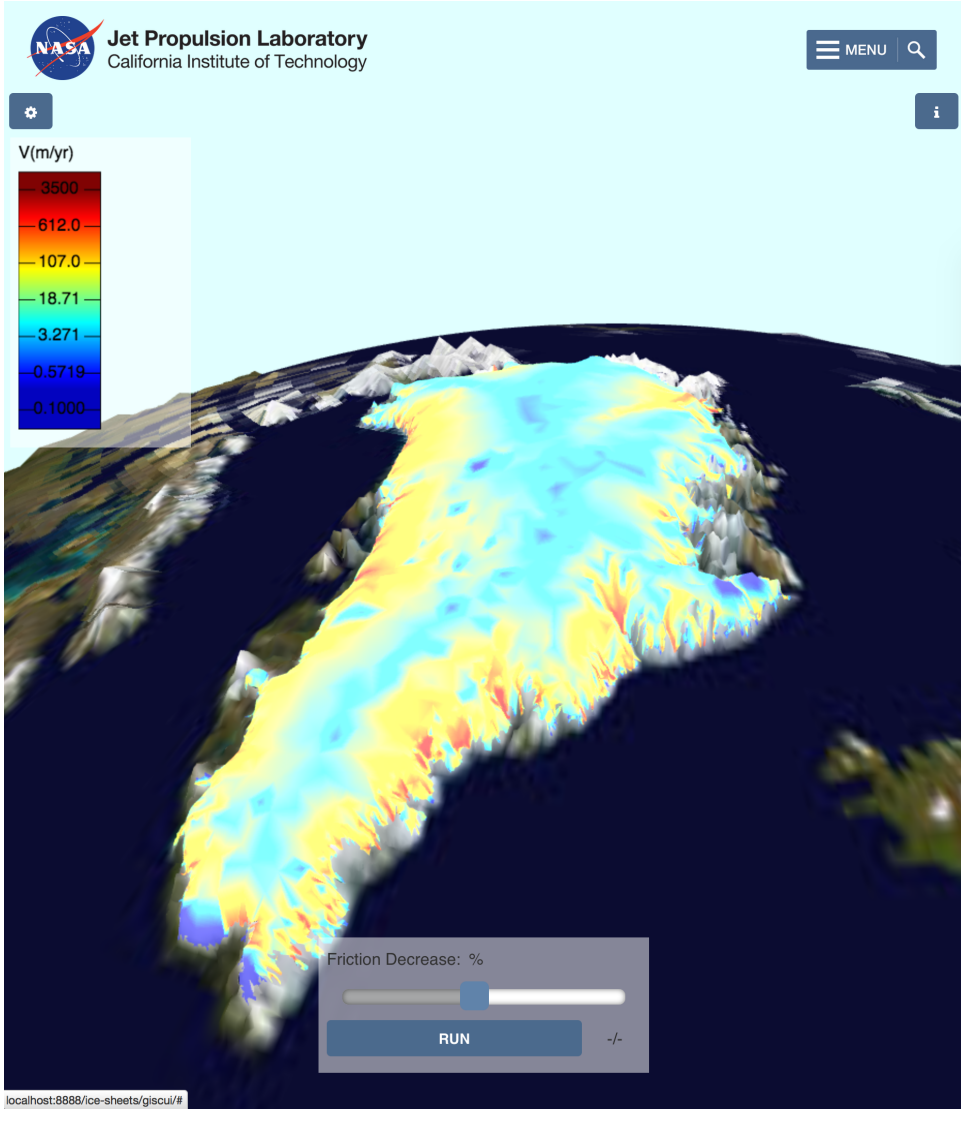

Figure 4. Greenland ISSM simulation on the Virtual Earth System Laboratory (ESL) (http://issm.jpl.nasa.gov/earthsystemlaboratorynew). This particular simulation allows for the introduction of user-driven friction anomalies (using a slider ranging from 5 to 100%) on the steady-state stress-balance velocities for the entire Greenland Ice Sheet. The computations (upon clicking of the RUN button) are carried out on the ISSM computational server (where the model inputs are uploaded, and from which the results are downloaded locally to the client' Web browser). The steady-state velocities are displayed for each value of the friction coefficient that the user chooses. The interactive rendering of the velocity field is done in 3D using the ISSM WebGl rendering engine. The results are overlaid on a semi-transparent topographical rendering of ETOPO5 data (see reference: National Geophysical Data Center (1988) for credits) and a background geotiff image from the Blue Marble: Land Surface, Shallow Water and Shaded Topography project (see reference: NASA Goddard Space Flight Center, Reto Stockli for credits).

References

Adhikari, S. and Marshall, S. J.: Improvements to shear-deformational models of glacier dynamics through a longitudinal stress factor, J. Glaciol., 57, 1003–1016, 2011.

Adhikari, S., Ivins, E. R., and Larour, E.: ISSM-SESAW v1. 0: mesh-based computation of gravitationally consistent sea-level and geodetic signatures caused by cryosphere and climate driven mass change, Geoscientific Model Development, 9, 1087–1109, doi:10.5194/gmd-9-1087-2016, 2016.

Bindschadler, R., Nowicki, S., Abe-Ouchi, A., Aschwanden, A., Choi, H., Fastook, J., Granzow, G., Greve, R., Gutowski, G., Herzfeld, U., Jackson, C., Johnson, J., Khroulev, C., Levermann, A., Lipscomb, W., Martin, M., Morlighem, M., Parizek, B., Pollard, D., Price, S., Ren, D., Saito, F.and Sato, T., Seddik, H., Seroussi, H., Takahashi, K., Walker, R., and Wang, W.: Ice-Sheet Model Sensitivities to Environmental Forcing and Their Use in Projecting Future Sea-Level (The SeaRISE Project), J. Glaciol., 59, 195–224, doi:10.3189/2013JoG12J125, 2013.

ECMA International: ECMAScript 2016 Language Specification, http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf, 2016.

Eyring, V., Bony, S., Meehl, G. A., Senior, C. A., Stevens, B., Stouffer, R. J., and Taylor, K. E.: Overview of the Coupled Model Intercomparison Project Phase 6 (CMIP6) experimental design and organization, Geoscientific Model Development, 9, 1937–1958, doi:10.5194/gmd-9-1937-2016, http://www.geosci-model-dev.net/9/1937/2016/, 2016.

Gropp, W., Lusk, E., Doss, N., and Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard, Parallel Computing, 22, 789–828, 1996.

Gropp, W. D. and Lusk, E.: User's Guide for `mpich`, a Portable Implementation of MPI, Mathematics and Computer Science Division, Argonne National Laboratory, aNL-96/6, 1996.

Inc, A.: Amazon Elastic Compute Cloud (Amazon EC2), Amazon Inc., http://aws.amazon.com/ec2/#pricing, http://aws.amazon.com/ec2/#pricing, 2008.

Jenkins: Jenkins, open source automation server, https://jenkins.io, 2016.

Larour, E. and Schlegel, N.: On ISSM and leveraging the Cloud towards faster quantification of the uncertainty in ice-sheet mass balance projections., Computers and GeoSciences, submitted, 2016.

Larour, E., Seroussi, H., Morlighem, M., and Rignot, E.: Continental scale, high order, high spatial resolution, ice sheet modeling using the Ice Sheet System Model (ISSM), J. Geophys. Res., 117, 1–20, doi:10.1029/2011JF002140, 2012.

Larour, E., Cheng, D., Perez, G., Quinn, J., Morlighem, M., Duong, B., Nguyen, L., Petrie, K., Harounian, S., Halkides, D., and Hayes, W.: Virtual Earth System Laboratory Website, http://vesl.jpl.nasa.gov, 2016.

MacAyeal, D.: Binge/Purge oscillations of the Laurentide ice-sheet as a cause of the North-Atlantic's Heinrich events, Paleoceanography, 8, 775–784, 1993.

Market, O.: Fastcgi: A high-performance web server interface, Technical white paper, 1996.

Molod, A., Takacs, L., Suarez, M., and Bacmeister, J.: Development of the GEOS-5 atmospheric general circulation model: evolution from MERRA to MERRA2, Geoscientific Model Development, 8, 1339–1356, doi:10.5194/gmd-8-1339-2015, http://www.geosci-model-dev.net/8/1339/2015/, 2015.

Morlighem, M., Seroussi, H., Larour, E., and Rignot, E.: Inversion of basal friction in Antarctica using exact and incomplete adjoints of a higher-order model, J. Geophys. Res., 118, 1746–1753, doi:10.1002/jgrf.20125, 2013.

NASA Goddard Space Flight Center, Reto Stockli: NASA Goddard Space Flight Center Image by Reto Stöckli (land surface, shallow water, clouds). Enhancements by Robert Simmon (ocean color, compositing, 3D globes, animation). Data and technical support: MODIS Land Group; MODIS Science Data Support Team; MODIS Atmosphere Group; MODIS Ocean Group Additional data: USGS EROS Data Center (topography); USGS Terrestrial Remote Sensing Flagstaff Field Center (Antarctica); Defense Meteorological Satellite Program (city lights).

National Geophysical Data Center: Data Announcement 88-MGG-02, Digital relief of the Surface of the Earth. NOAA, National Geophysical Data Center, Boulder, Colorado, 1988, 1988.

Nowicki, S., Payne, A., Larour, E., Seroussi, H., Goelzer, H., Lipscomb, W., Gregory, J., Abe-Ouchi, A., and Shepherd, A.: Ice Sheet Model Intercomparison Project (ISMIP6) contribution to CMIP6 , Geosci. Model Dev., 9, 4521–4545, doi:10.5194/gmd-9-4521-2016, 2016.

OpenMP Architecture Review Board: OpenMP Application Program Interface Version 4.5, http://www.openmp.org/mp-documents/openmp-4.5.pdf, 2015.

Pattyn, F.: GRANTISM: An ExcelTM model for Greenland and Antarctica ice-sheet response to climate changes, Comp. Geosci., 32, 316–325, 2005.

Rignot, E.: PALSAR studies of ice sheet motion in Antarctica, in: ALOS PI Symposium, Nov. 3-7 2008, 2008.

Taylor, K., Stouffer, R., and Meehl, G.: An Overview of CMIP5 and the experiment design, Bull. Amer. Math. Soc., 93, 485–498, doi:10.1175/BAMS-D-11-00094.1, 2012.

Taylor, K. E., Stouffer, R. J., and Meehl, G. A.: A Summary of the CMIP5 Experiment Design, 2009.

Vaughan, G. V., Elliston, B., Tromey, T., and Taylor, I. L.: GNU Autoconf, Automake, and Libtool, Pearson Education, http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1578701902, 2000.

World Wide Web Consortium: HTML 4.0 Specification – W3C Recommendation – Conformance: requirements and recommendations, https://www.w3.org/TR/REC-html40-971218/conform.html, 1997.

Zakai, A.: Emscripten: An LLVM-to-JavaScript Compiler, in: Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion, OOPSLA '11, pp. 301–312, ACM, New York, NY, USA, doi:10.1145/2048147.2048224, http://doi.acm.org/10.1145/2048147.2048224, 2011.