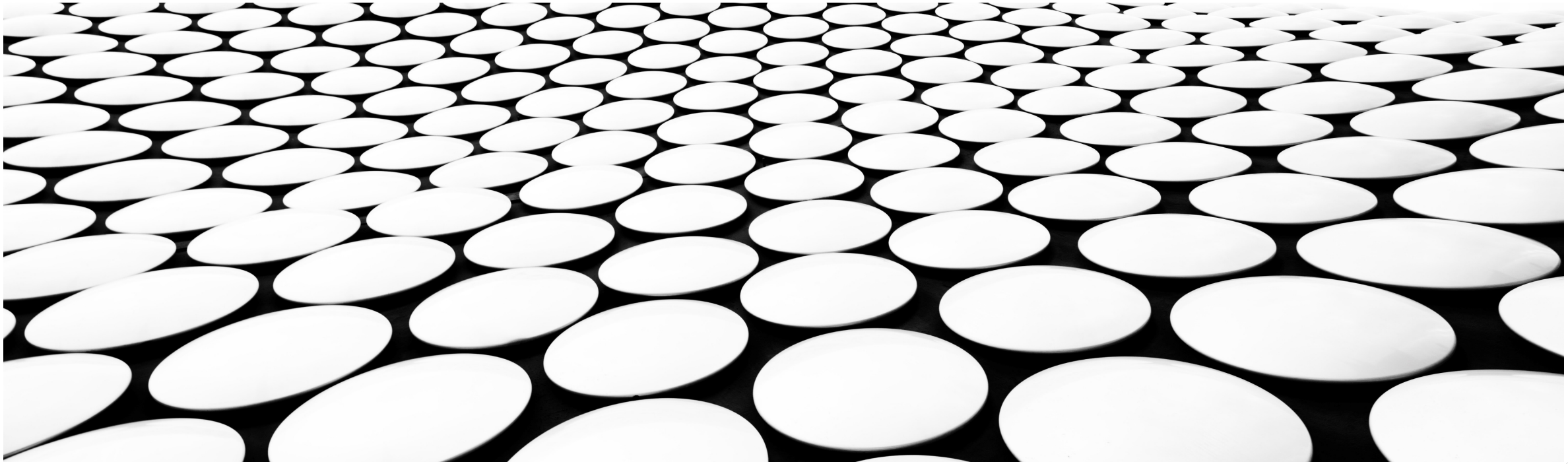
IMPLEMENTATION AND TESTING OF THE COMMUNITY MULTISCALE AIR QUALITY (CMAQ) MODEL ON THE CLOUD

ELIZABETH ADAMS, CARLIE COATS, CHRISTOS EFSTATHIOU AND SARAVANAN ARUNACHALAM, UNC INSTITUTE FOR THE ENVIRONMENT

MARK REED AND ROBERT ZELT, UNC INFORMATION TECHNOLOGY SERVICES



CMAQ ON THE "CLOUD"

Goal: Implement and test the Community Multiscale Air Quality (CMAQ) model for on-demand access to a large remote pool of computing and data resources offered through commercial "cloud" vendors

- Address computing, software, and data issues collectively and make recommendations
- Develop summary of best practices for the CMAS User Community

Computing: Commercial cloud computing platforms vs local computing infrastructure

- Amazon Web Services (AWS)
- Microsoft Azure

Software: Use of Virtual Machines, container or csh scripted native build approach to replicate existing system and software environment, so researchers avoid configuring from scratch (time & difficulty)

Data problem: Large volumes of data can be quickly shared and processed in the cloud, saving time from downloading locally and keeping redundant copies

DEFINING THE STEPS

Approach:

- Establish a set of benchmark cases representative of the CMAQ community needs
- Review and enhance existing methods to connect to share input data, store and distribute output
- Keep in mind other factors such as visualization tools, debugging, etc.

This work aims to:

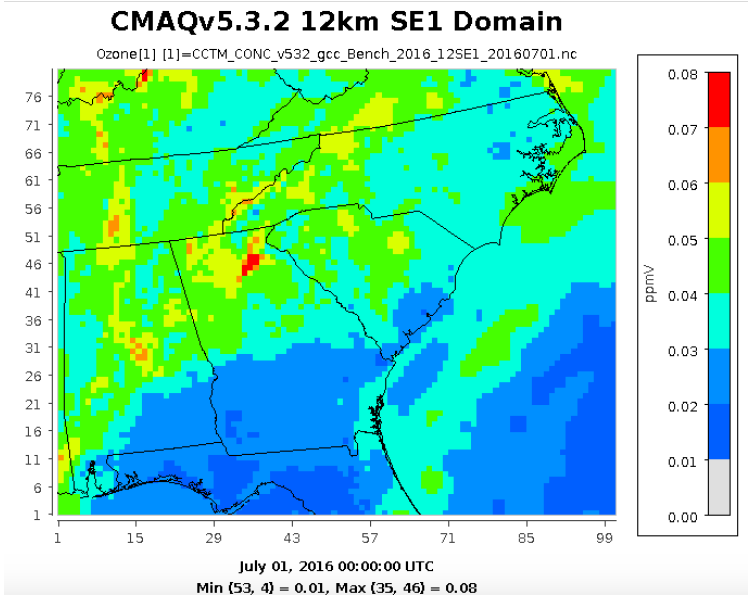
- Develop methods to build, install, and run CMAQ as a Singularity container, Docker container or scripted native build
- Test performance and scalability on single node cloud computing environments
- Test multi-node high performance computing on the cloud using ***Amazon ParallelCluster*** and ***Microsoft Cyclecloud***
  - Ways to provision and manage HPC workloads using any scheduler to run MPI jobs
- Develop recommendations on provisioning resources, accurately forecasting CMAQ model run time with optimal configuration, storage requirements, and ultimately create reliable cost estimates for performing CMAQ simulations on the cloud

THE CMAQ BENCHMARK SUITE

Hardware configurations depend on the domain size, grid resolution, the number of variables and layers saved to the output

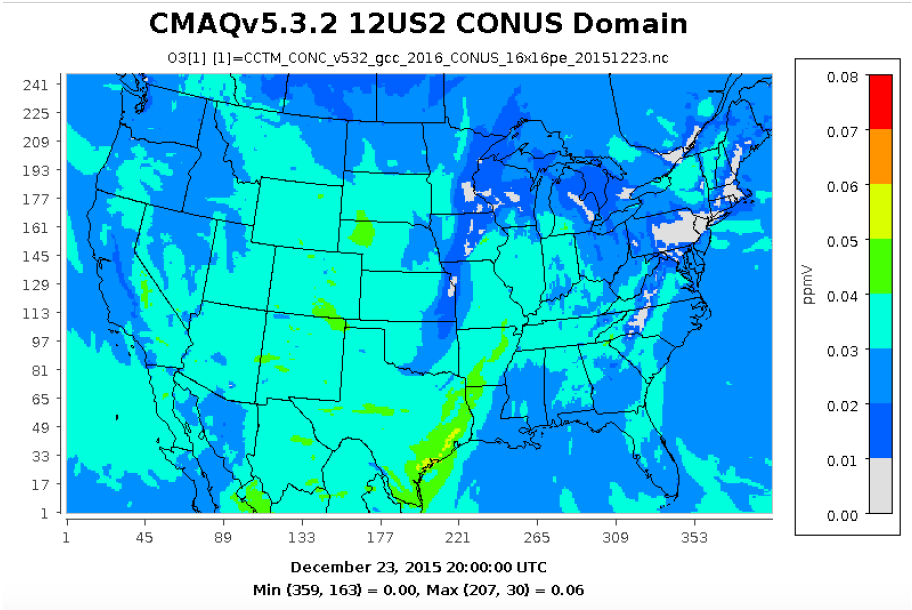
Typical requirements for two different 2-Day Benchmark Cases, both using a 12x12-km horizontal grid resolution

- Domain 1: Distributed 12SE1 Benchmark Case (NCOLS= 100, NROWS=80, NLAYS=35)
- Domain 2: CONUS (NCOLS= 396, NROWS=246, NLAYS=35)



Storage Requirements

Input:
23GB
Output:
15GB (full)
1.2GB (12 vars, 1 layer)



Storage Requirements

Input:
44GB
Output:
172GB (full)
17.7 GB (12vars, 1 layer)

THE CMAQ PLATFORM CONFIGURATION EVOLUTION

- Base-compiler and libraries used (We assumed the standard CentOS7)
- MPI version and its major version number (and underlying base-compiler) for the build
- "vanilla" CMAQ, or Decoupled Direct Method (CMAQ-DDM), or the Integrated Source Apportionment Method (CMAQ-ISAM)
- Compiler options: We assumed medium memory model (see https://cjcoats.github.io/ioapi/AVAIL.html#medium) and both debug and optimized, with default hardware-model (which means SSE4.2).

Native System Build

- OS:CentOS
- GCC 8.3.1
- OpenMPI 3.1.4
- NETCDF 4.7.1
- NETCDF-Fortran 4.5.2
- IOAPI 3.2
- CMAQ v5.3.2

Container System Build

- OS:CentOS
- Bootstrap: docker
- GCC 9.2
- OpenMPI 3.1.4
- hdf5-1.10.5
- pnetcdf-1.11.2
- NETCDF 4.7.1
- NETCDF-Fortran 4.5.2
- IOAPI 3.2
- CMAQ v5.3.2

pCluster System Build

- OS:CentOS
- GCC 8.3.1
- OpenMPI 3.1.4
- NETCDF 4.7.1
- NETCDF-Fortran 4.5.2
- IOAPI 3.2
- CMAQ v5.3.2

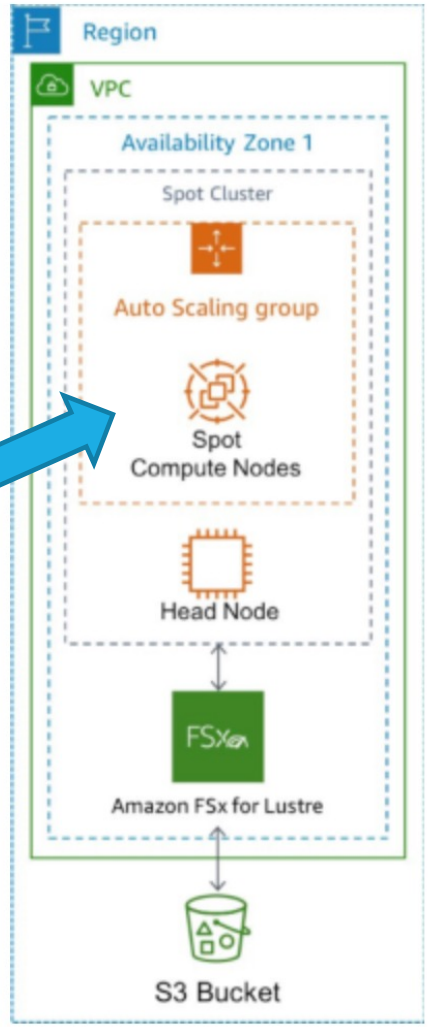
Cycle Cloud System Build

- OS:CentOS
- Bootstrap: docker
- GCC 9.2
- OpenMPI 4.0.5
- NETCDF 4.7.1
- NETCDF-Fortran 4.5.2
- IOAPI 3.2
- CMAQ v5.3.2

HARDWARE AND STORAGE OPTIONS ON AWS

- Simulation benchmarks : CMAQ
- C5n Compute Instances with 100 Gbps Networking
- Parallel Cluster can be configured to use any of the instances as compute nodes and to reduce pricing, can be reserved as spot instances (reducing cost).
- C5n.18xlarge instances support Elastic Fabric Adapter (EFA) to run tightly-coupled HPC applications at scale on AWS.

Instance Name	vCPUs	RAM	EBS Bandwidth	Network Bandwidth
c5n.large	2	5.25 GiB	Up to 3.5 Gbps	Up to 25 Gbps
c5n.xlarge	4	10.5 GiB	Up to 3.5 Gbps	Up to 25 Gbps
c5n.2xlarge	8	21 GiB	Up to 3.5 Gbps	Up to 25 Gbps
c5n.4xlarge	16	42 GiB	3.5 Gbps	Up to 25 Gbps
c5n.9xlarge	36	96 GiB	7 Gbps	50 Gbps
c5n.18xlarge	72	192 GiB	14 Gbps	100 Gbps



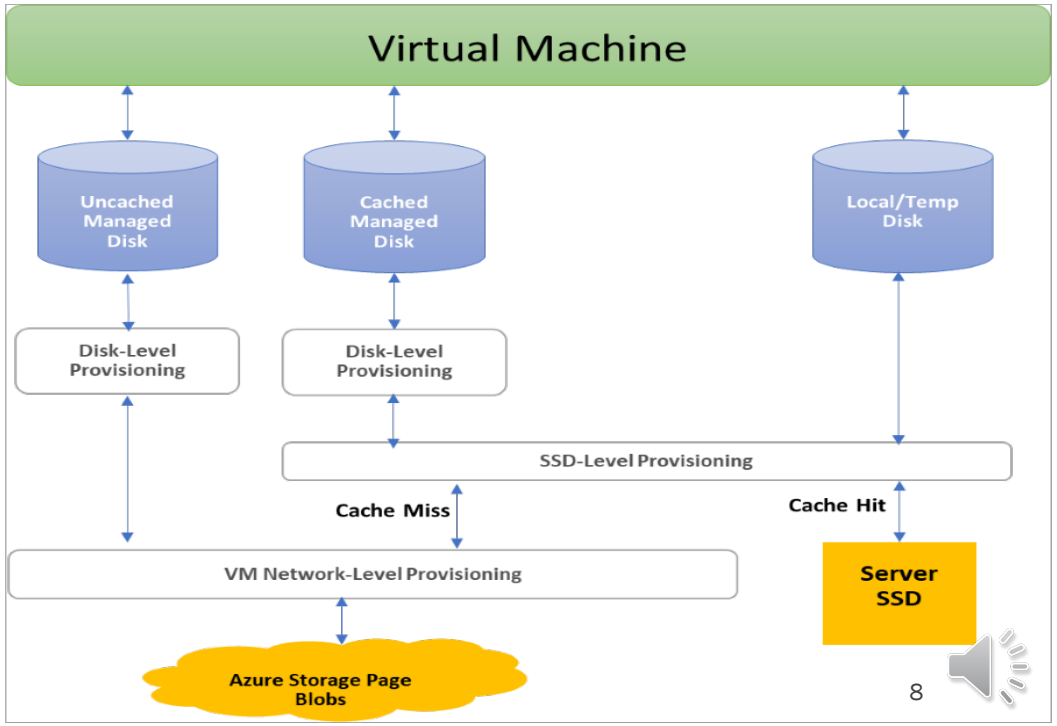
HARDWARE OPTIONS ON AZURE - VIRTUAL MACHINES

Type	Sizes	Description
General purpose	B, Dsv3, Dv3, Dasv4, Dav4, DSv2, Dv2, Av2, DC, DCv2, Dv4, Dsv4, Ddv4, Ddsv4	Balanced CPU-to-memory ratio. Ideal for testing and development, small to medium databases, and low to medium traffic web servers.
Compute optimized	F, Fs, Fsv2, FX	High CPU-to-memory ratio. Good for medium traffic web servers, network appliances, batch processes, and application servers.
Memory optimized	Esv3, Ev3, Easv4, Eav4, Ev4, Esv4, Edv4, Edsv4, Mv2, M, DSv2, Dv2	High memory-to-CPU ratio. Great for relational database servers, medium to large caches, and in-memory analytics.
Storage optimized	Lsv2	High disk throughput and IO ideal for Big Data, SQL, NoSQL databases, data warehousing and large transactional databases.
GPU	NC, NCv2, NCv3, NCasT4_v3, ND, NDv2, NV, NVv3, NVv4	Specialized virtual machines targeted for heavy graphic rendering and video editing, as well as model training and inferencing (ND) with deep learning. Available with single or multiple GPUs.
High performance compute	HB, HBv2, HBv3, HC, H	The fastest and most powerful CPU virtual machines with optional high-throughput network interfaces (RDMA).

HARDWARE OPTIONS ON AZURE – STORAGE AND NETWORK

- Provisioning takes place at different levels, user must be aware of the cost of options vs benefit
- HBv3-series VMs are optimized for HPC applications. HBv3 VMs feature up to 120 AMD EPYC™ 7003-series (Milan) CPU cores, 448 GB of RAM, and no hyperthreading. HBv3-series VMs also provide 350 GB/sec of memory bandwidth, up to 32 MB of L3 cache per core, up to 7 GB/s of block device SSD performance, and clock frequencies up to 3.675 GHz.
- All HBv3-series VMs feature 200 Gb/sec HDR InfiniBand from NVIDIA Networking to enable supercomputer-scale MPI workloads. These VMs are connected in a non-blocking fat tree for optimized and consistent RDMA performance. The HDR InfiniBand fabric also supports Adaptive Routing and the Dynamic Connected Transport (DCT, in additional to standard RC and UD transports). These features enhance application performance, scalability, and consistency, and their usage is strongly recommended.
- Benchmark tests on the Azure environment were performed using two different managed disk options, the standard SSD tier E20 and the premium SSD tier P40
- A number of factors determine the final storage cost to the user and therefore it is advised to use the pricing calculator to better estimate individual scenarios (https://azure.microsoft.com/en-us/pricing/calculator/)
- Indicatively, the cost for 512 GB E20 standard SSD starts about $38.60 per month while the P40 premium equivalent is nearly double that of the standard SSD cost

Size	vCPU	Processor	Memory (GiB)	Memory BW (GB/s)	Base CPU Freq. (GHz)	RDMA Perf (Gb/s)
Standard_HB120rs_v3	120	AMD EPYC 7V13	448	350	2.45	200
Standard_HB120-96rs_v3	96	AMD EPYC 7V13	448	350	2.45	200
Standard_HB120-64rs_v3	64	AMD EPYC 7V13	448	350	2.45	200
Standard_HB120-32rs_v3	32	AMD EPYC 7V13	448	350	2.45	200
Standard_HB120-16rs_v3	16	AMD EPYC 7V13	448	350	2.45	200



CONTAINERIZING CMAQ AND SCRIPTED NATIVE BUILD OF LIBRARIES AND CODE

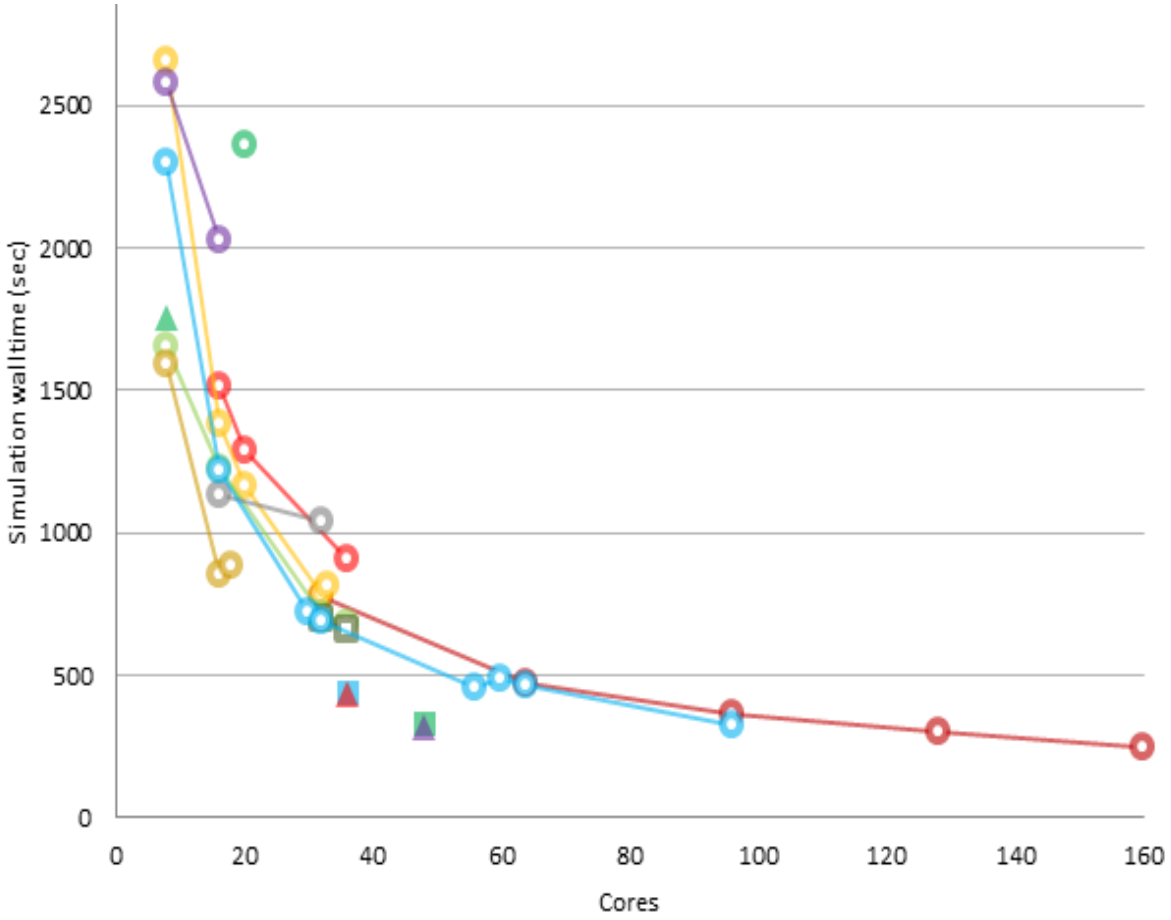
Benefits:

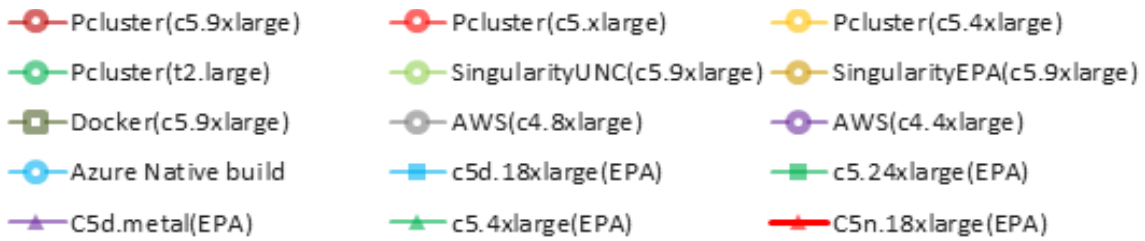
Layered approach allows the user to rebuild the final layer without touching the underlying ones

Also allows for any required updates or changes to be tracked using GitHub

- Definition file #1 contains instructions for downloading and installing GNU Compiler Collection (GCC 9.2), HDF5 (hdf5-1.10.5), netCDF-C (netcdf-c-4.7.1), netCDF-Fortran (netcdf-fortran-4.5.2), netCDF-CXX (netcdf-cxx4-4.3.1), OpenMPI (openmpi-3.1.4), and Parallel netCDF (pnetcdf-1.11.2), and takes build time of approximately 25 minutes
- Definition file #2 contains instructions for building a tagged release of I/O API, with build time of 2 minutes
- Definition file #3 contains instructions for building CMAQv5.3.2
- The initial method for building the SingularityCE container layers was provided by the EPA for an earlier version of CMAQ.
- The three scripts used to create the definition files were modified to run as C-shell scripts to create "Native Builds" of the software
- All definition files and scripts use wget and configure to obtain and build libraries, git pull is used to obtain CMAQ, with bldit.cctm and a modified run script to build and run CMAQ
- Once the software is loaded to an instance, it can be saved and reused as an Amazon Machine Instance (AMI) or Azure Custom Image

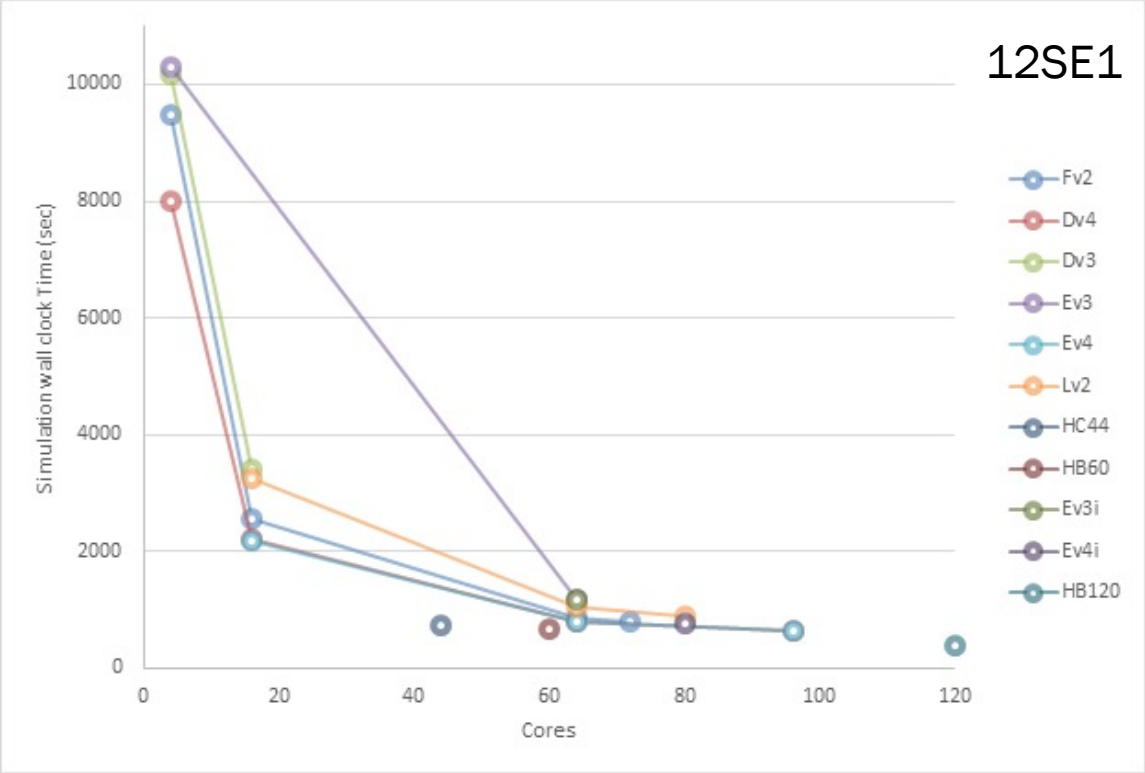
RESULTS FROM NATIVE AND CONTAINERIZED SIMULATIONS (AWS + AZURE) 12SE1

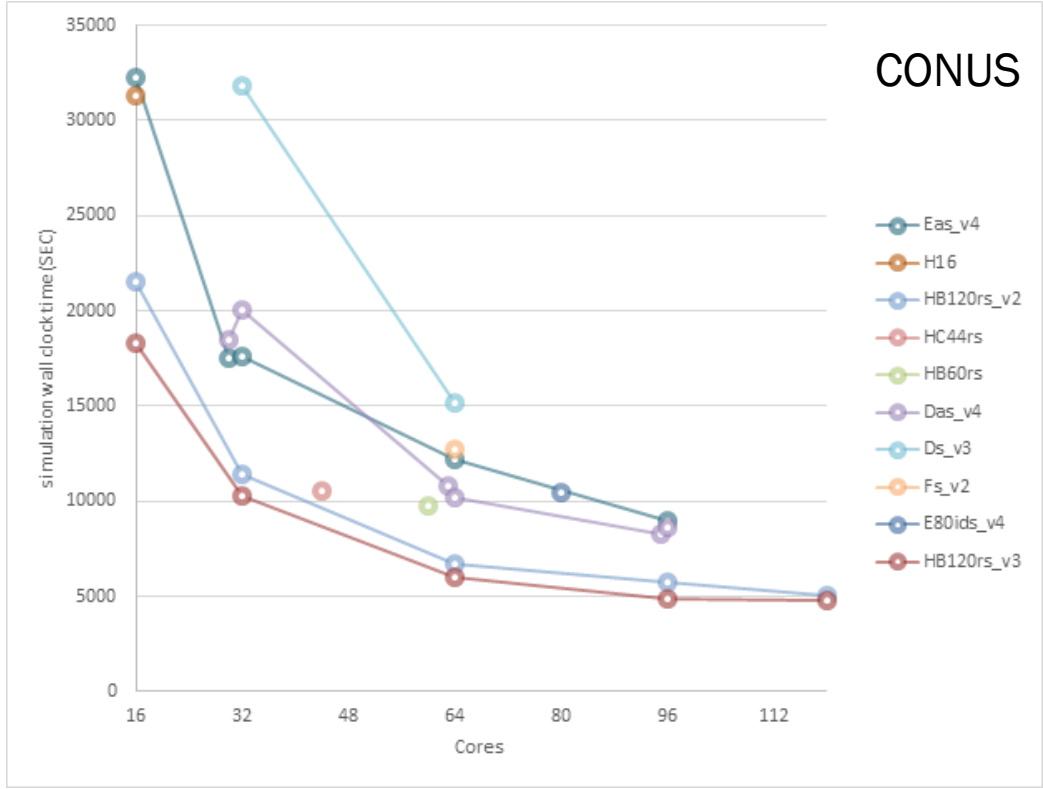




1. Running on a single node on the AWS the performance is limited by the number of CPUs available per node, and whether you are using virtual cpus (hyperthreading turned on) or not.
2. Successfully ran using containers on an AWS single node, but not on the Parallel Cluster
3. Successfully used Parallel Cluster to compile and run a native build, which allows you to use more CPUs, but the smaller domain of the benchmark limits the ability to fully utilize additional CPUs.

RESULTS FROM CONTAINERIZED SIMULATIONS ON AZURE





1. Similarly to AWS, when running single node on Azure the performance is limited by the number of CPUs available and the processor generation and speed
2. Successfully ran using containers on Azure single nodes, memory limits the use of VMs with less than 16cores for CONUS
3. Testing different tiers revealed best performance obtained using the latest generation of HPC tier VMs

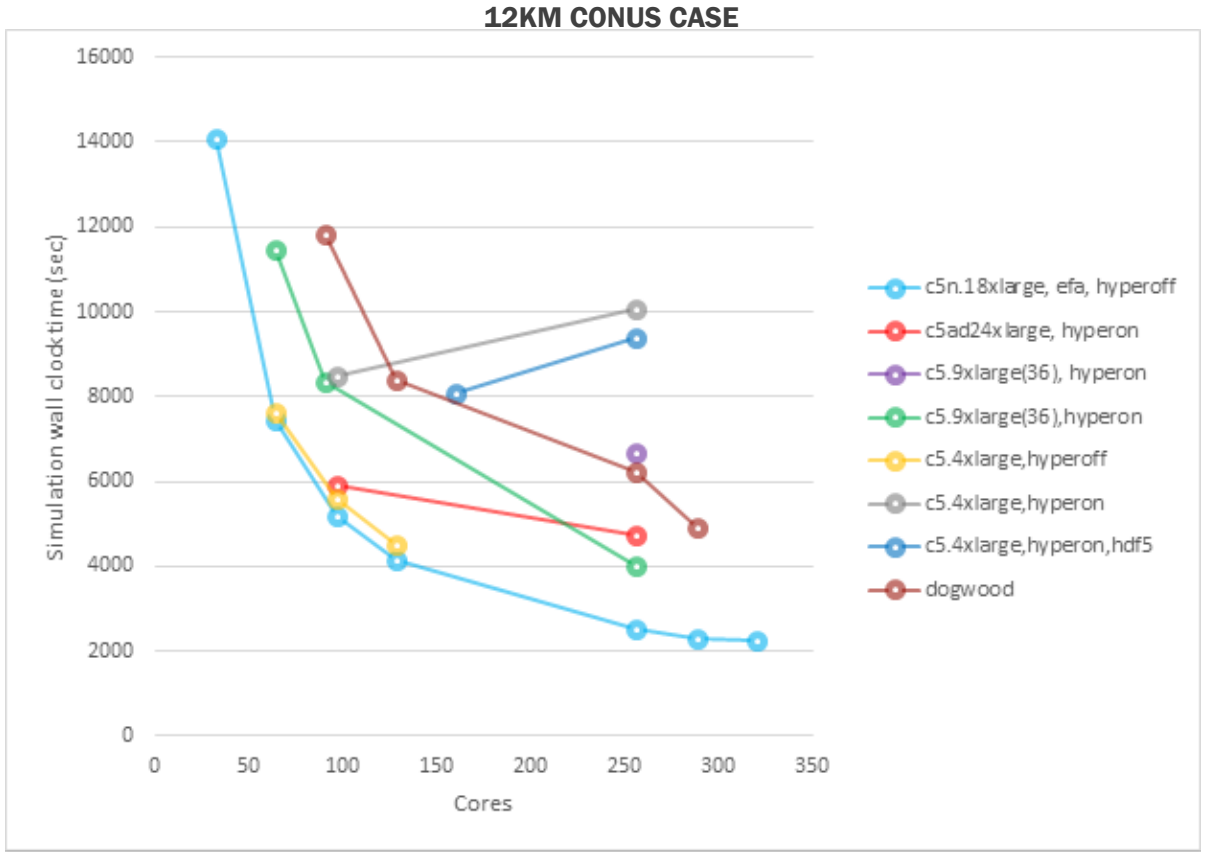
INSIGHTS FROM CONTAINERIZED SIMULATIONS ON AZURE – COSTS*

		Regular SSD option						Premium SSD option					
VM	VM cost per hour ($)	Cores	Day 1 (sec)	Day 2 (sec)	Total Time	CONUS VM Cost ($)	Annual Run VM cost ($)	Cores	Day 1 (sec)	Day 2 (sec)	Total Time	CONUS Cost*	Annual Run VM cost ($)
E16as_v4	1.01	16	16723	15516	32239	9.0	1655	-	-	-	-	-	-
H16	1.09	16	15795	15470	31265	9.5	1732	-	-	-	-	-	-
D32as_v4	1.54	32	10678	9397	20075	8.6	1572	-	-	-	-	-	-
D32as_v4	1.54	30	9584	8938	18522	7.9	1450	-	-	-	-	-	-
D32s_v3	1.54	32	16714	15146	31860	13.6	2494	-	-	-	-	-	-
E32as_v4	2.02	32	9606	7960	17566	9.9	1804	-	-	-	-	-	-
E32as_v4	2.02	30	9180	8334	17514	9.8	1798	-	-	-	-	-	-
HC44rs	3.16	44	5589	4911	10500	9.2	1687	-	-	-	-	-	-
HB60rs	2.28	60	5069	4702	9771	6.2	1132	-	-	-	-	-	-
D64as_v4	3.07	64	5399	4790	10190	8.7	1590	64	5607	4835	10442	8.9	1630
D64as_v4	3.07	63	5813	5022	10835	9.2	1691	-	-	-	-	-	-
D64s_v3	3.07	64	8144	7025	15170	12.9	2367	-	-	-	-	-	-
E64as_v4	4.03	64	6595	5626	12221	13.7	2504	-	-	-	-	-	-
F64s_v2	2.7	64	6789	5942	12731	9.5	1747	-	-	-	-	-	-
E80ids_v4	5.76	80	5606	4878	10483	16.8	3070	80	5715	4977	10693	17.1	3131
D96as_v4	4.61	96	4523	4071	8594	11.0	2014	96	4439	3968	8407	10.8	1970
D96as_v4	4.61	95	4328	3916	8245	10.6	1932	-	-	-	-	-	-
E96as_v4	6.05	96	4790	4156	8946	15.0	2751	-	-	-	-	-	-
HB120rs_v2	3.6	120	2611	2472	5083	5.1	930	120	2692	2380	5072	5.1	928
HB120rs_v2	3.6	119	2697	2419	5117	5.1	936	119	2651	2375	5026	5.0	920
HB120rs_v2	3.6	96	3000	2776	5776	5.8	1057	96	2969	2742	5711	5.7	1045
HB120rs_v2	3.6	64	3568	3153	6721	6.7	1230	64	3646	3191	6837	6.8	1251
HB120rs_v2	3.6	32	5967	5473	11441	11.4	2094	32	5916	5329	11245	11.2	2058
HB120rs_v2	3.6	16	11203	10316	21520	21.5	3938	16	11842	10579	22421	22.4	4103
HB120rs_v3	3.6	120	2547	2274	4821	4.8	882	120	2447	2187	4634	4.6	848
HB120rs_v3	3.6	96	2619	2252	4871	4.9	891	96	2584	2217	4800	4.8	878
HB120rs_v3	3.6	64	3242	2747	5989	6.0	1096	64	3121	2722	5843	5.8	1069
HB120rs_v3	3.6	32	5387	4874	10261	10.3	1878	32	5388	4897	10285	10.3	1882
HB120rs_v3	3.6	16	9588	8683	18272	18.3	3344	16	9503	8682	18185	18.2	3328

1. Latest generations of HPC tier VMs are the fastest and most cost-effective solution in Azure
2. Storage option did not have a significant impact on the speed to justify the cost (this needs to be evaluated by user for larger simulations)
3. Leaving 1-2 cores free helped slightly speed up the simulations
4. Availability and pricing can make older generations and general compute tier appealing for smaller scale simulations

*Note that the costs refer only to VM use, are indicative, and for comparison purposes

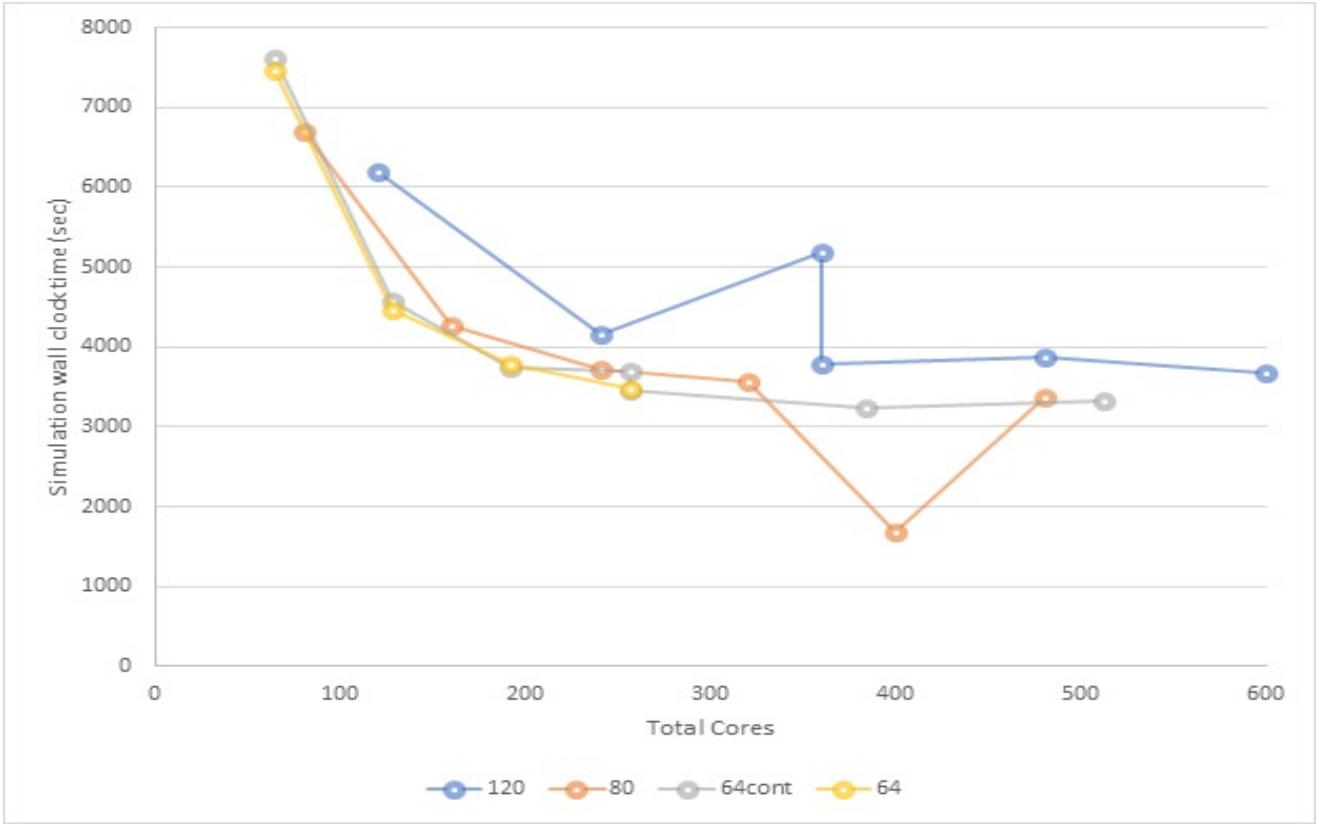
MULTI-NODE SIMULATION RESULTS: AWS PARALLELCLUSTER NATIVE BUILD



1. Using the Parallel Cluster allows you to quickly iterate testing using different EC2 instances, choosing enhanced networking support, including enabling elastic fabric adapter (efa) with hyper-threading turned on or off and modifying the underlying filesystem, quickly copying input data from the S3 Bucket, and choosing spot or reserved instances using a configuration file.
2. The best performance for the 12km CONUS domain was obtained using the c5n.18xlarge instance with efa, with hyperthreading turned off, using the lustre file system, and using spot instance pricing.

MULTI-NODE SIMULATION RESULTS: AZURE CYCLECLOUD

Cores	Time(s)	Nodes	Speedup	Procs/ Node
120	6207	1	1.000	120
240	4165	2	1.490	
360	5182	3	1.198	
360	3781	3	1.642	
480	3870	4	1.604	
600	3686	5	1.684	
80	6692	1	1.000	80
160	4280	2	1.668	
240	3717	3	2.035	
320	3575	4	2.066	
400	1693	5	2.208	
480	3366	6	2.359	
64	7624	1	1.000	64cont
128	4572	2	1.668	
192	3748	3	2.035	
256	3690	4	2.066	
256	3453	4	2.208	
384	3232	6	2.359	
512	3335	8	2.286	
64	7474	1	1.000	64
128	4469	2	1.672	
192	3790	3	1.972	
256	3470	4	2.154	



1. The best performance for the 12km CONUS domain on Azure was obtained using the most recent HPC tier VMs
2. Using fewer Procs/Node can result to substantially faster simulations
3. The container version of CMAQ performed similarly to the native Cyclecloud setup

LESSONS LEARNED

Single-node simulations

- Container-based approach performs and scales well in both systems
- More modern nodes have speed advantages and better performance/price
- Relatively easy to provision and similar learning curves between the system
- In general, best suited for smaller scale simulations, although the HPC tier performed very well for CONUS

Multi-node simulations

- Not as easy to setup as the single-node environment – learning curve steeper in both systems with Azure need more provisioning work
- Networking and MPI added level of difficulty for containerized approach
- Both approaches scale well, breaking down the simulations in smaller chunks can help speed up (done easier in SLURM than single node setup)

CONCLUSIONS

- Benefits to working on the cloud
  - Quickly set-up and run - no waiting in job queue
  - Compute resource and storage is reliable, resizable, and quickly configurable
  - Flexible pricing – on-demand versus spot-pricing
  - AWS Parallel Cluster and Azure CycleCloud provides access to hundreds or thousands of compute cores, a job scheduler for keeping them fed, a shared file system that's tuned for throughput or IOPS (or both), updated compilers and libraries, and a fast network.
  - AWS Parallel Cluster "*infrastructure as code*" - single shell command can create a complex HPC cluster, ***and*** a Lustre file system, ***and*** a visualization studio, Azure Cycle Cloud provides some of this functionality but requires additional steps and assistance from UNC IT Support to provision.
  - AWS Share input and output data on s3 buckets with very fast download speeds using S3 API
  - AWS Parallel Cluster and Azure CycleCloud were difficult to configure run MPI across nodes within a container, but both supported building the libraries and compiling CMAQv5.3.2 natively and running using OpenMPI and the SLURM Job Scheduler.
  - Cost comparisons between similarly configured AWS Parallel Cluster and Azure Cycle Cloud couldn't be made directly, due to differences in how the clusters were provisioned and tested.

FUTURE WORK

- Phase 1: Create Minimum Viable Product (MVP) for CMAS Community to be released by Spring 2022
  - Ability to build (native build and not in container) and install CCTM (from CMAQ v5.3.3) on the cloud for the ConUS case (2016 12km case) on 2 environments - AWS (pCluster) and Azure (Cycle Cloud) for 2 days
  - Post-processing (combine, etc.) to reduce the data volume for potential future egress out of AWS/Azure
  - Ability to access input datasets from EPA's S3-bucket using 2016 platform
  - Include robust testing (using varying # of cores and hardware configs as may be available)
  - Documentation of entire procedure (both install and data retrieval), with tutorial for the user community to emulate for their own applications
- Phase 2: Expanded capabilities [Timeline: TBD]
  - Move CMAQ to package management software (Yum or other)
  - Change scripts from C-shell to Bash
  - Enable running containers on both AWS pCluster and Cycle Cloud
  - Access data from CMAS Data Warehouse [e.g., EQUATES datasets or other] *[See Foley et al Presentation in Model Applications Session on Tuesday]*
  - Add additional post-processing tools (e.g., AMET, VERDI or other python tools) *[See Session on Python for CMAQ by Kim and Henderson at this Conference on Fri]*

REFERENCES

- Singularity Container Build Method and Documentation by Dr. Carlie Coats
  - https://cjcoats.github.io/CMAQ-singularity/
- Singularity Container Build Method by Ed Anderson
- Amazon AWS Parallel Cluster
  - https://aws.amazon.com/hpc/parallelcluster/
  - https://aws.amazon.com/blogs/storage/building-an-hpc-cluster-with-aws-parallelcluster-and-amazon-fsx-for-lustre/
- Azure Cycle Cloud
  - https://azure.microsoft.com/en-us/features/azure-cyclecloud/#overview
  - https://docs.microsoft.com/en-us/azure/cyclecloud/how-to/hb-hc-best-practices?view=cyclecloud-8#centos-76-hpc-marketplace-image

ACKNOWLEDGMENTS

- The U.S. EPA, through its Office of Research and Development, partially funded and collaborated in the research described here under Contract EP-W-16-014 to the Institute for the Environment at the University of North Carolina at Chapel Hill.
- John McGee, UNC-ITS and Microsoft Azure for Cloud Credits
- We thank the following staff at the EPA for several inspiring and helpful discussions and contributions to this work
  - Kristen Foley
  - Fahim Sidi
  - Steve Fine
  - Ed Anderson
  - Tom Pierce