Harvard John A. Paulson School of Engineering and Applied Sciences

WHERE SCIENCE AND ENGINEERING CONVERGE

optRBC:

Optimal Solutions in Rayleigh-Benard Convection

Group 8: Katrina Gonzalez, Michael Neuder, Jack Scudder

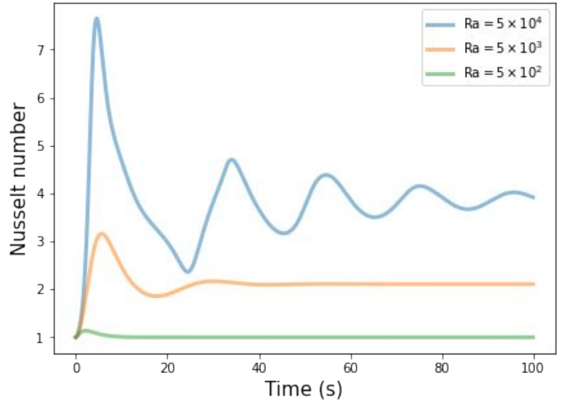
10 May 2021

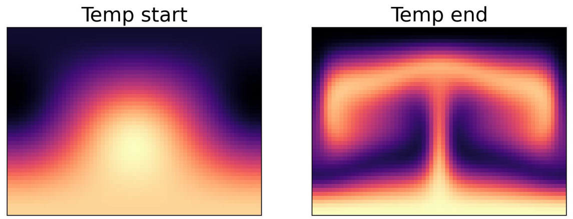
Agenda

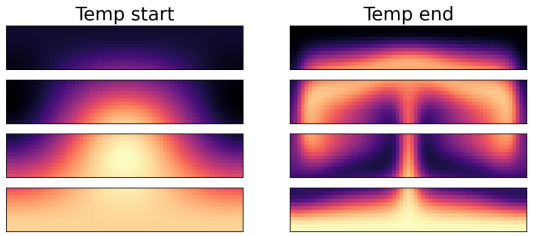
- Project Overview
- Contributions
  - OpenMP Parallelization
  - MPI Parallelization
- Additional Experiments/Tests
  - FFTW
  - Academic Cluster
- Interesting Challenges
- Closing Thoughts

Project Overview

- Rayleigh Benard convection (RBC), discretized to solve Oberbeck-Boussinesq PDEs.
- Study of RBC at high Rayleigh numbers important for turbulent fluid simulation.
- **Application:** HPC Problem: Compute-intensive, originally fully serial.
- **Levels of Parallelism Implemented:**
  - Fine Grained - Loops/Procedure Level (OpenMP)
  - Coarse Grained - Task Level discretization at scale (MPI)
- Experimented with multithreaded FFT, one-sided FFT
- OpenMP: 6x max speedup, MPI: 2x max speedup (8 core AWS VM)

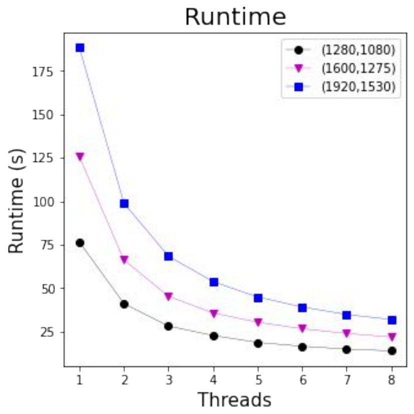


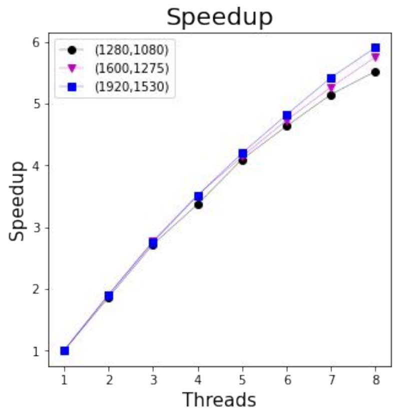


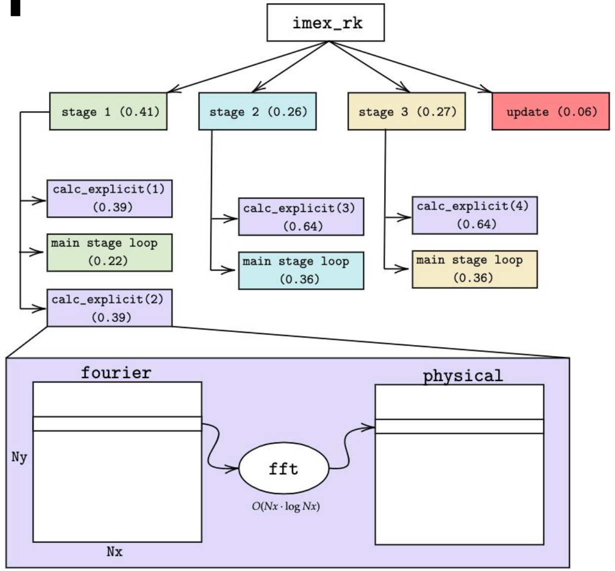


Contributions: OpenMP Parallelism

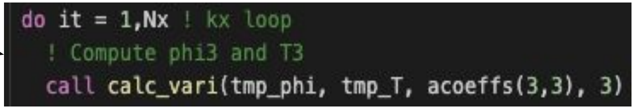
- From profiling, majority of time in **`calc_explicit`**, which executes 8*Ny fast fourier transforms per time step.
- OpenMP: Parallelized 16 loops.
- Strong scaling shown below (**`t2.2xlarge`** on AWS).
- **Insight**: Subroutines that access global variables need to be refactored to have globals injected as arguments.





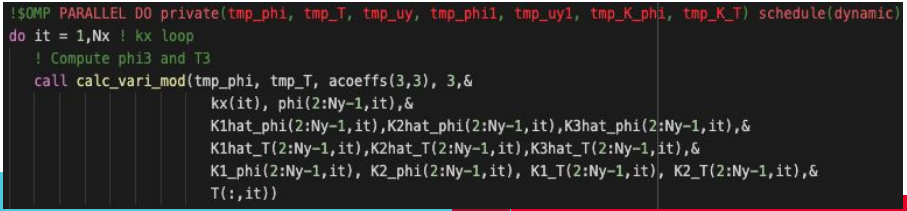


Before



```
do it = 1,Nx ! kx loop
  ! Compute phi3 and T3
  call calc_vari(tmp_phi, tmp_T, acoeffs(3,3), 3)
```

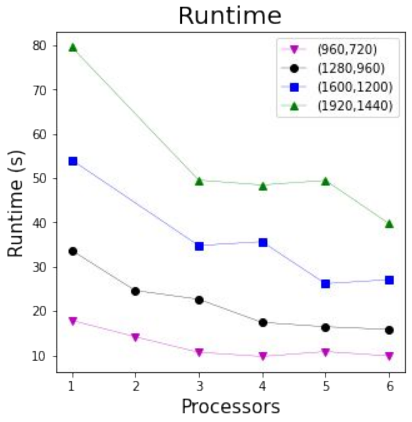
After

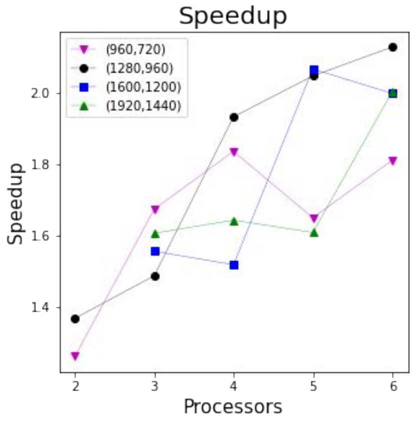


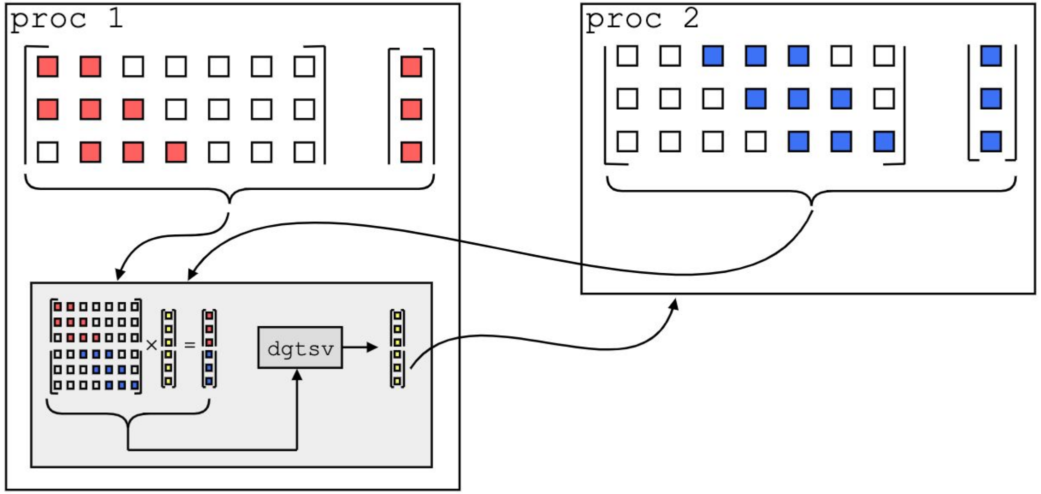
```
!$OMP PARALLEL DO private(tmp_phi, tmp_T, tmp_uy, tmp_phi1, tmp_uy1, tmp_K_phi, tmp_K_T) schedule(dynamic)
do it = 1,Nx ! kx loop
  ! Compute phi3 and T3
  call calc_vari_mod(tmp_phi, tmp_T, acoeffs(3,3), 3,&
                     kx(it), phi(2:Ny-1,it),&
                     K1hat_phi(2:Ny-1,it),K2hat_phi(2:Ny-1,it),K3hat_phi(2:Ny-1,it),&
                     K1hat_T(2:Ny-1,it),K2hat_T(2:Ny-1,it),K3hat_T(2:Ny-1,it),&
                     K1_phi(2:Ny-1,it), K2_phi(2:Ny-1,it), K1_T(2:Ny-1,it), K2_T(2:Ny-1,it),&
                     T(:,it))
```

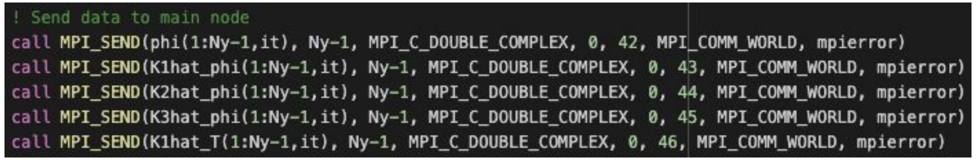
Contributions: MPI Parallelism

- Implicit updates require solving tridiagonal matrix equations.
- Strong scaling shown below (**`t2.2xlarge`** on AWS).
- **Insight**: This implementation requires O(Nx*Ny) data to be sent per time step.





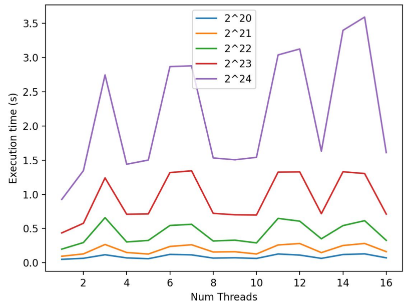


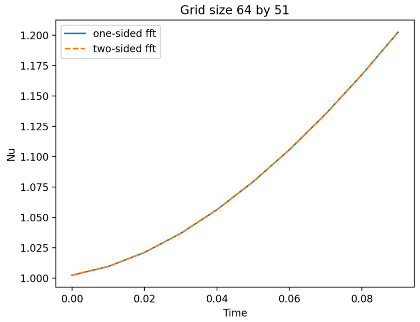


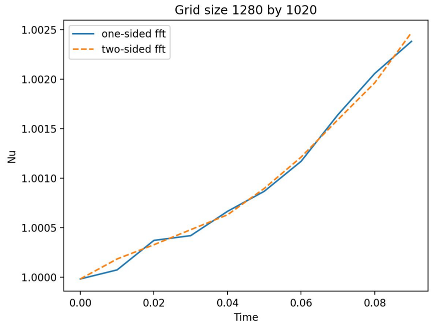
```
! Send data to main node
call MPI_SEND(phi(1:Ny-1,it), Ny-1, MPI_C_DOUBLE_COMPLEX, 0, 42, MPI_COMM_WORLD, mpierror)
call MPI_SEND(K1hat_phi(1:Ny-1,it), Ny-1, MPI_C_DOUBLE_COMPLEX, 0, 43, MPI_COMM_WORLD, mpierror)
call MPI_SEND(K2hat_phi(1:Ny-1,it), Ny-1, MPI_C_DOUBLE_COMPLEX, 0, 44, MPI_COMM_WORLD, mpierror)
call MPI_SEND(K3hat_phi(1:Ny-1,it), Ny-1, MPI_C_DOUBLE_COMPLEX, 0, 45, MPI_COMM_WORLD, mpierror)
call MPI_SEND(K1hat_T(1:Ny-1,it), Ny-1, MPI_C_DOUBLE_COMPLEX, 0, 46, MPI_COMM_WORLD, mpierror)
```

Experiments: Multithreaded, One-sided FFT

- **Multithreaded FFT**
  - Option to use multiple threads to execute FFT in FFTW library.
  - **Insight:** Toy code demonstrated lack of speedup for problem size.
- **One-sided FFT**
  - Physical problem, initial and final fields (temperature, velocity) should be real.
  - Exploit Hermitian conjugate symmetry for algorithmic speedup.
  - **Insight:** With existing implementation, errors in Nusselt number proportional to **`Nx`**. Does not seem to accumulate in time.



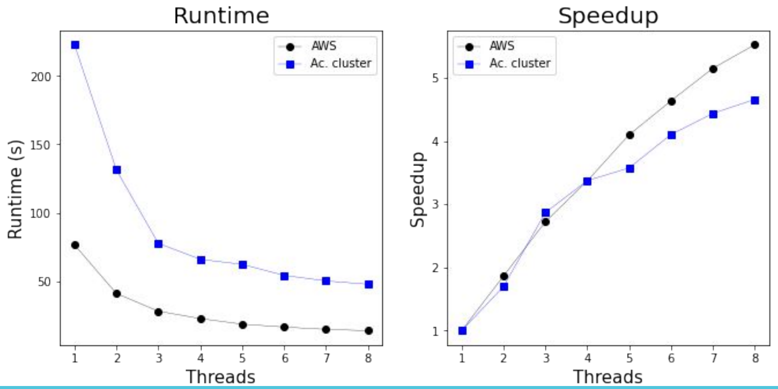




Harvard John A. Paulson School of Engineering and Applied Sciences

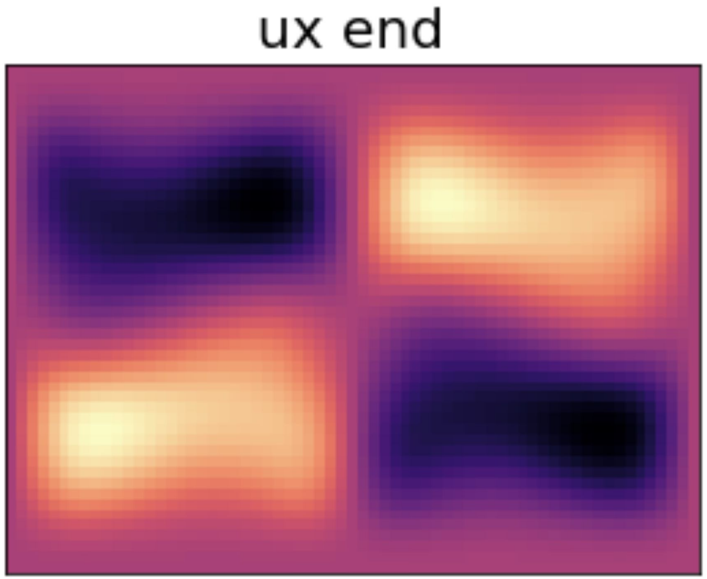
Measurements: Cloud vs. Cluster

- Cloud (AWS)
  - For benchmarking, using **t2.2xlarge** on AWS
- Academic Cluster
  - Allocated 16 cores, 4GB per CPU
- **Insights:** Found much more consistent performance using AWS vs. Academic cluster, likely due to greater resource sharing and different hardware. This may have implications for general performance replicability on the Academic Cluster.
- Attempted different provisioning, plots show best run



Interesting Challenges

- Working in Fortran.
- Diving into mathematically rich existing code base; Fourier vs. physical space, reading loops in **`Nx`**.
- Difficult to parallelize loops in the MPI implementation that had send and receives (multiple threads).



Derivative in the x-direction at the end of integration.

Conclusion

- Parallelized existing fluid dynamics codebase.
- Used several approaches:
  - Multithreaded FFT
  - One-sided FFT
  - OpenMP
  - MPI
- Ran performance analysis in AWS and the academic cluster.
- Learned a ton about Fortran and scientific computing… and had fun!
- Single big takeaway: it is possible to parallelize scientific computing code, *but it is not easy, especially at scale.*

A FEW
OF MY
FAVORITE
THINGS

Thanks!

- To our fellow classmates in the course
- To the TFs for supporting us through labs and office hours and teaching us everything we know about AWS
- And to Dr. Sondak for letting us work on his code and meeting with us regularly to work through challenges!