Lessons Learned from Massively Parallel Model of Ventilator Splitting

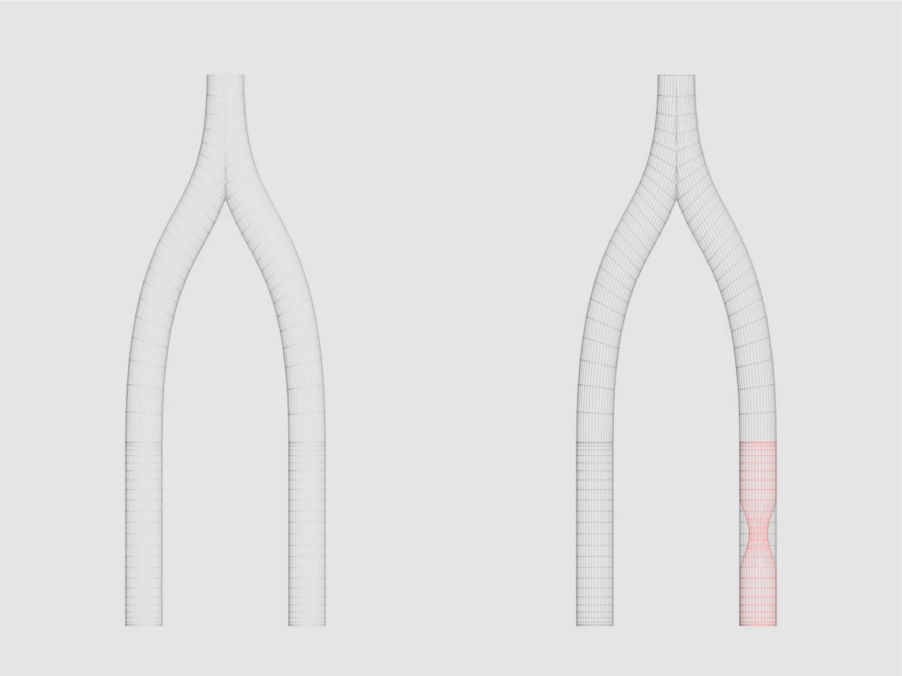
Michael Kaplan, Charles Kneifel, Victor Orlikowski, James Dorff, Mike Newton, Andy Howard, Don Shin, Muath Bishawi, Simbarashe Chidyagwai, Peter Balogh, Amanda Randles

Need for an expansion of ventilator capacity

- Ventilators are vital equipment for assistance with respiration
- A shortfall of 45,000-160,000 ventilators with the U.S. was predicted for the ongoing pandemic
- Duke researchers paired up with restor3D to develop a safe, efficacious ventilator splitting technology

What was needed?

- Tuning to select optimal restrictor for specific patients
- Computational fluid dynamics model to capture airflow
- Validation of fluid model



Courtesy of Liam Krauss, LLNL

HPC Consortium: Microsoft Azure

- Coupled with Microsoft and Duke Office of Information Technology
- Worked with Microsoft Azure
- Need for minimal time-to-solution
- Result: ~800,000 compute hours over one weekend

Lesson 1: Low overhead tools

Using low overhead tools such as Matlab's Simscape for initial modeling enabled rapid acquisition of baseline intuition needed for the design of the large-scale study

Lesson 2: Straight-forward parallelization

Relying on an embarrassingly parallel framework allowed us to match to a dynamic cloud-based environment that best facilitated the required large-scale parameter sweep.

Lesson 3: Problem-oriented approach

By configuring the cloud architecture through a problem-oriented approach instead of manipulating the problem to fit the constraints of the platform, we were able to rapidly deploy the model

Lesson 4: Mirroring the testing environment

By mimicking the feasibility testing environment, we were able to re-use scripts and rapidly implement the model.

Lesson 5: Identify hidden interdependencies

Beyond initial performance testing, full-scale tests deploying the model helped to identify hidden dependencies causing severe performance degradation

Lesson 6: Interaction

A cross-platform mobile app with a selection-based UI and **pre-calculated** values maximizes usability in various hospital settings, while minimizing user error.