Robust Platform for Scientific Computing: Python

Yung-Yu Chen

yyc@solvcon.net

Python Hsinchu User Group

September 24, 2013

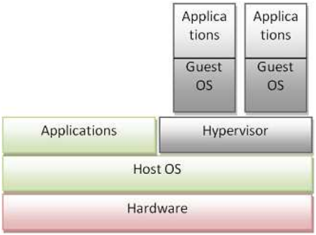
Topics

Supercomputing

- Specialized hardware for specific applications.
- Speed is the number 1 objective.

- Top 500 list (`http://top500.org/lists/2013/06/`):
  1. Tianhe-2: 3.12M cores, 33.86 Pflops (Peta floating-point operations per second). Equips Xeon E5-2692 and Xeon-Phi 31S1P.
  2. Titan (Cray XK7): 0.56M cores, 17.59 Pflops. Equips Opteron 6274 and NVIDIA K20x.
  3. Sequoia (IBM BlueGene/Q): 1.57M cores, 17.17 Pflops. Equips Power PQC 16C.

Cloud Computing



- Provide elastic computing power through virtualization technology.
  - Programs are run not on real hardware, but the virtualized systems.
  - Users can dynamically allocate resources including computing nodes and cores, memory, storage, etc.
- Pay-as-you-go allows everyone to solve significantly large problems.

But This Is Really We Are Using

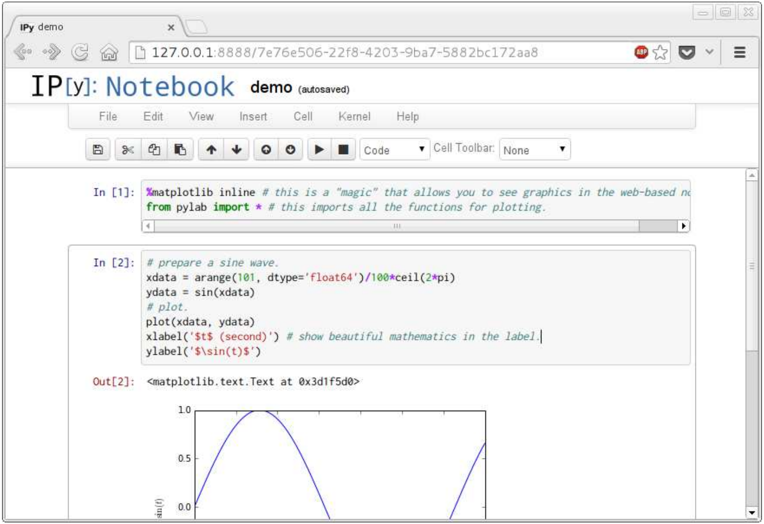
Programming Platform Matters

- You need a programming language, or several programming languages, to process, simulate, analyze, and exhibit the data of your problems.
    - It doesn't matter what supercomputers you are using.
    - It doesn't matter what cloud services you are using.
    - It doesn't matter if you are running the code on your laptop.
- You need to be proficient at a powerful programming language.
- And Python is that language.

What Python Can Do?

- An everyday tool:
  `python -c 'import math; print math.factorial(10)'`.
- An established ecosystem for reproducible analysis.
  - A rich collection of scientific tools: NumPy, SciPy, Matplotlib, Pandas, iPython, etc.
- A popular platform for web programming.
  - Even include a web server:
    `python -m SimpleHTTPServer`.
- A platform that allows you to do anything.
  - The Python Package Index (`https://pypi.python.org/pypi`) now has 34,924 packages.

A Rich Interactive Environment



Before Started

- The Python official site contains comprehensive information:
  - Table of contents of the documentation: `http://docs.python.org/2/`.
  - The best reference to the standard library: `http://docs.python.org/2/library/index.html`.
  - Read *The Python Tutorial* (`http://docs.python.org/2/tutorial/index.html`).
- Start with Python 2.
- A good self-training material is *Learn Python the Hard Way* (`http://learnpythonthehardway.org/book/`).

Installation

- Use Anaconda.
  - `https://store.continuum.io/cshop/anaconda/`.
  - It provides everything at once and can be easily updated.
- NOT recommended for scientific users:
  - Download from Python website.
  - Use your OS's package managers (apt-get, yum, ports, etc.)
- Why not?
  - They require expertise in Python and take long time.
  - They contains outdated packages.

*This suggestion is specific to scientific users.

Two Types of Programming

- Compiled vs interactive.
  - Low-level platform is built upon pre-compiled executables, like the Python runtime and the underneath libraries.
  - In a high-level interactive environment, we can type just several commands or press buttons to do complex computing.
- Python glues the low-level parts together, and expose them to the high-level.
- To configure and build the low-level platform needs a lot of efforts.
  - Products like Anaconda do that for us.

Categories of Scientific Python Tools

- Programming.
  - NumPy, Cython, iPython.
- Algorithms.
  - SciPy, SciKits.
- Visualization.
  - Matplotlib, VTK.
- Applications.
  - Pandas, NLTK, networkx, PyMOL, Pyomo, yt, ..., etc.

NumPy

- NumPy (`http://numpy.scipy.org/`) provides basic multi-dimensional array support.
- Array-oriented programming is the foundation to scientific computing.
- It provides basic facilities such as linear algebra and Fourier transform.

Let's see the demo.

Cython

- Cython (`http://cython.org/`) is a superset of the Python programming language.
- It speeds up Python code to be comparable of C.
- It provides interfaces for Python to use low-level C code or libraries.

Cython Benchmark

```
import numpy as np
cimport numpy as cnp
def action():
    cdef cnp.ndarray[cnp.double_t, ndim=2] arr0 = np.empty([1000,1000], dtype='float64')
    arr0.fill(0)
    cdef cnp.ndarray[cnp.double_t, ndim=2] arr1 = np.empty([1000,1000], dtype='float64')
    arr1.fill(1)
    cdef int it = 1
    cdef int jt
    while it < 999:
        jt = 1
        while jt < 999:
            arr0[it, jt] += arr1[it-1, jt-1]
            arr0[it, jt] += arr1[it-1, jt  ]
            arr0[it, jt] += arr1[it-1, jt+1]
            arr0[it, jt] += arr1[it  , jt+1]
            arr0[it, jt] += arr1[it+1, jt+1]
            arr0[it, jt] += arr1[it+1, jt  ]
            arr0[it, jt] += arr1[it+1, jt-1]
            arr0[it, jt] += arr1[it  , jt-1]
            jt += 1
        it += 1
    assert 7968032 == arr0.sum()
```

Cython is 41 times faster than normal Python.

iPython Notebook

- iPython stands for interactive Python.
- It provides plain-text, GUI, and web interface.
- iPython notebook is its web interface.
  - Use `ipython notebook` to launch.
  - Useful for research notes, experimenting, and education.
  - Mathematical expressions are the first-class citizen.
  - You can store and share any ipython notebook, even online: `http://nbviewer.ipython.org/`.

Let's see more demo of it.

SciPy Library

- The term SciPy has many meanings:
  - The SciPy library (`http://docs.scipy.org/doc/scipy/reference/`); what I want to talk about here.
  - The SciPy ecosystem (`http://www.scipy.org/`); everything about Python for sciences.
  - The SciPy conference (`http://conference.scipy.org/`); in North America, Europe, and India.
  - The SciPy community; those who use Python for scientific research.

Let's see how it works in an ipython notebook.

It's about SOLVCON

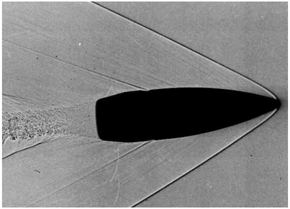
A solver constructor.

- Perform first-principle simulations for physical processes governed by conservation laws.
  - Usually formulated as hyperbolic partial differential equations (PDEs).
- Written in Python and with the performance hot-spot accelerated by C (or CUDA).
- Address high-performance computing (HPC) by mesh-based, array-oriented programming.

See `http://solvcon.net/` for detail.

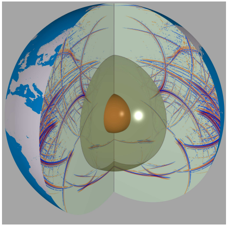
Conservation Laws Govern The World

Fluid mechanics, solid mechanics, electromagnetism, etc.

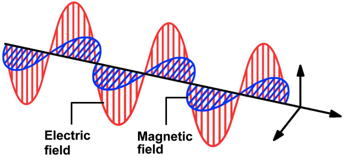


Supersonic flow.

Atmospheric flow.



Seismic waves.



Electromagnetic waves.

First-Order Hyperbolic PDEs

- The fore-mentioned problems share a common trait: Demanding time-accurate solutions of conservation laws.
- So this is what I want to solve:

$$
\begin{aligned}
&\frac{\partial u_m}{\partial t} + \sum_{\mu=1}^{3} \frac{\partial f_m^{(\mu)}(\mathbf{u})}{\partial x_\mu} = s_m(\mathbf{u}) \\
\Rightarrow\ &\boxed{\oint_{S(V)} \mathbf{h}_m(\mathbf{u}) \cdot d\mathbf{a} = \int_V s_m(\mathbf{u}) dv} \\
&m = 1, \ldots, M.
\end{aligned}
\tag{1}
$$

Challenges in Programming

Coding for first-principle simulators is difficult. Why?

1. Recall the math:

$$\frac{\partial u_m}{\partial t} + \sum_{\mu=1}^{3} \frac{\partial f_m^{(\mu)}(\mathbf{u})}{\partial x_\mu} = s_m(\mathbf{u}) \tag{1}$$

2. Various approaches to meshing and the associated data structures.
3. Parallel programming for HPC.
4. Data management and result analysis.

The CESE Method

- The space-time Conservation Element and Solution Element (CESE) method, developed by Chang at NASA Glenn.
  - Directly solves generic hyperbolic PDEs (Eq. (1)).
- Enable pluggable multi-physics in SOLVCON.
  - Compressible flows: $\mathbf{u} = (\rho, \rho v_1, \rho v_2, \rho v_3, \rho e)^t$.
  - Stress waves in solids: $\mathbf{u} = (v_1, v_2, v_3, \sigma_{11}, \sigma_{22}, \sigma_{33}, \sigma_{23}, \sigma_{13}, \sigma_{12})^t$.
  - Electromagnetic waves: $\mathbf{u} = (E_1, E_2, E_3, B_1, B_2, B_3)^t$.
  - Acoustics, shallow-water, viscoelasticity, etc.

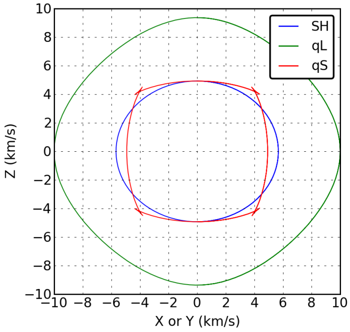
Chang (1995) Journal of Computational Physics 119(2):295–324
Chen (2011), Ph.D. Dissertation

What Is SOLVCON

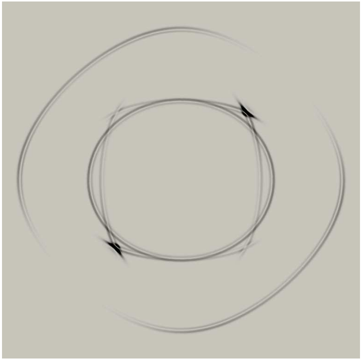
- A Python-based software framework for constructing time-accurate solvers of conservation laws for any physical processes.
- SOLVCON uses the CESE method.
  - Unstructured meshes of mixed elements are used in two- or three-dimensional space.
  - Message-passing is built into the framework for parallel computing.

Application: Stress Wave in Solids

- Beryl: Anisotropic crystal of hexagonal symmetry.



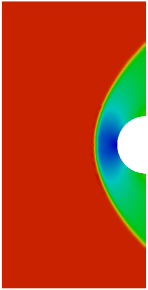
Exact solution of group velocity

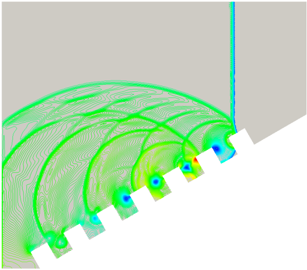


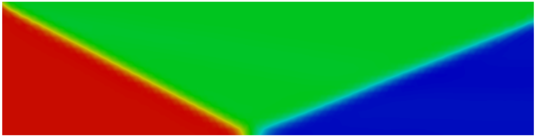
Simulated result

Yang et al. (2011) J. Vib. Acoust. 133(2): 021001

Application: Supersonic Flows



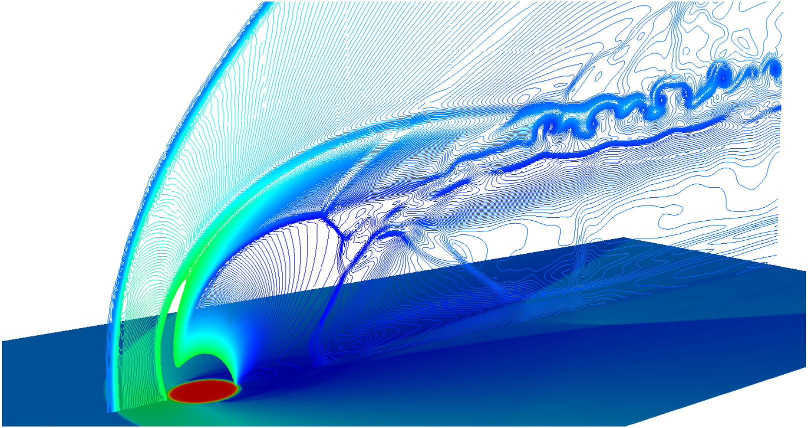




- 2D cases:
  - Flow over a cylinder.
  - Oblique shock by a ramp.
  - Moving shock climbing a ramp.
  - Moving shock diffraction by a step.
  - Moving shock past dust layer.
  - Reflection of oblique shock.
  - Implosion.
- 3D cases:
  - Sod's shock tube.
  - Flow over sphere.
  - Jet in cross flow.

Jet in Supersonic Cross Flow

66 million elements are used in the simulation.



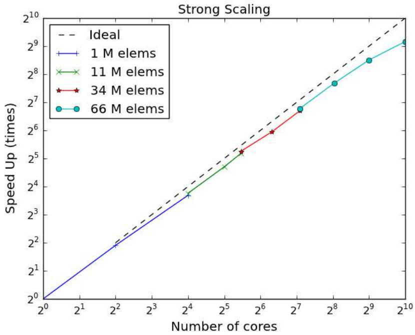
Density

Runtime Benchmark

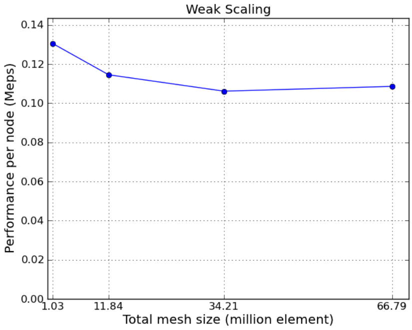
- Benchmark with hybrid parallel computing.
  - MPI across nodes; pthread within a node.
  - Run on Glenn@OSC: 4 cores/node with 10Gbps IB.
- Performance in million elements per second (Meps).

Number of cells (M)		1	11	34	66
Perf. (Meps)	1 core	0.035	–	–	–
	4 cores	0.13	–	–	–
	16 cores	0.45	0.47	–	–
	32 cores	–	0.91	–	–
	44 cores	–	1.26	1.33	–
	80 cores	–	–	2.16	–
	136 cores	–	–	3.61	3.82
	264 cores	–	–	–	7.17
	512 cores	–	–	–	12.7
	1024 cores	–	–	–	20.0

Scaling



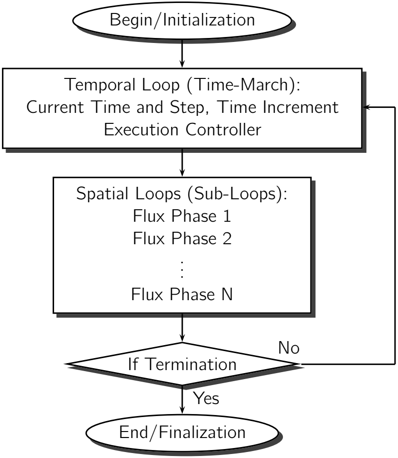
Fix Overall Mesh Size



Fix Per-Node Mesh Size

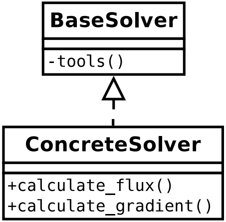
Two-Loop Structure of PDE Solvers

- The basic execution flow of SOLVCON:
  - Temporal loop for temporal (or pseudo-temporal) integration.
  - Spatial loops iterate over elements.
- The structure is general to all PDE solvers.



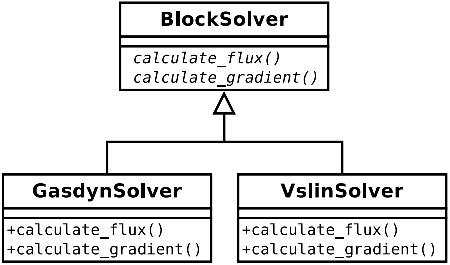
Solver Kernel for Spatial Loops

- A solver kernel is a Python class.
- The base class implements utility methods for spatial loops.
- The algorithms directly work with the mesh look-up tables.
- The concrete solver implements real algorithms, in C, or other fast languages.



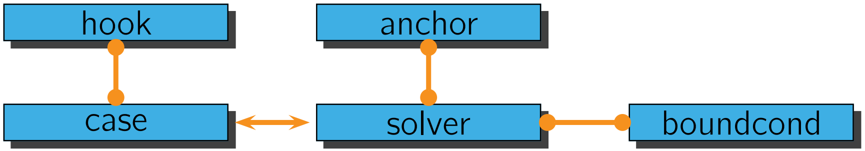
Inheritance for Multi-Physics

- For a multi-physics algorithm, like the CESE method, a class hierarchy can be designed to host multiple physical processes.
- The physical processes are segregated.



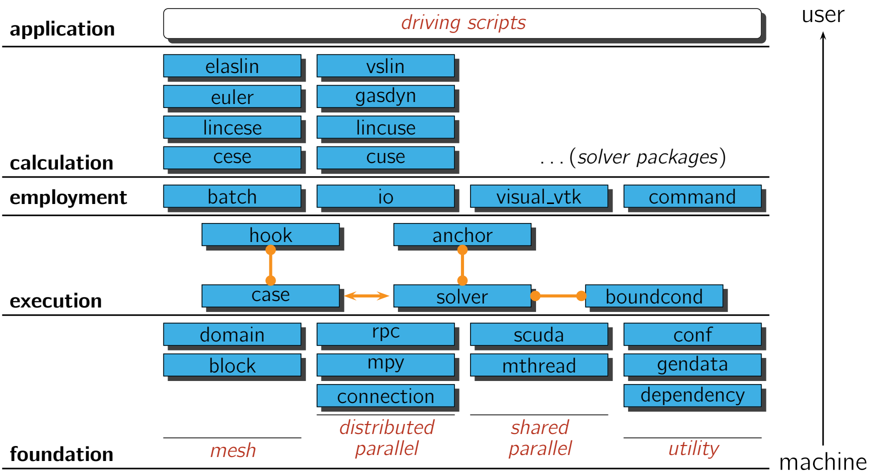
Temporal Loop and Call-Back

- A standalone class hierarchy (`Case`) is designed to host the temporal loop.

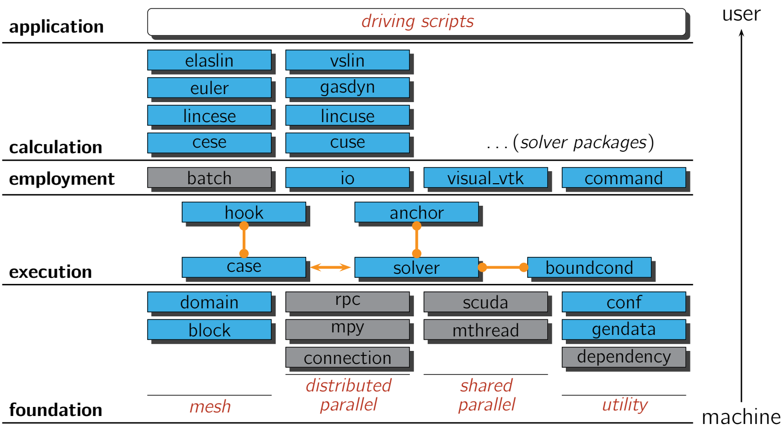


- `Hook` and `Anchor` are call-back objects for `Case` and `Solver`, respectively.
  - Supplement of main algorithms.
  - Lazy initialization.
  - Facilitating parallel computing and in-situ analysis.

Overall Design of SOLVCON

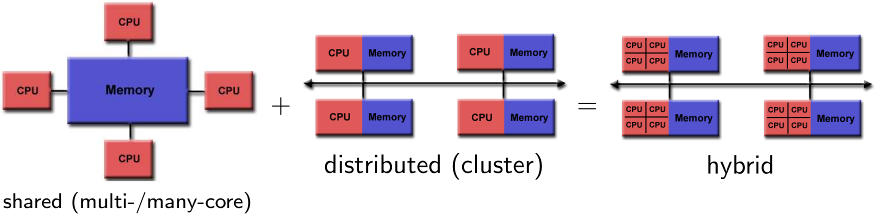


Renovation under Construction



- Simplify the architecture: Rely more on mpi4py, OpenMP, etc.
- Use Cython instead of ctypes for maintainability.

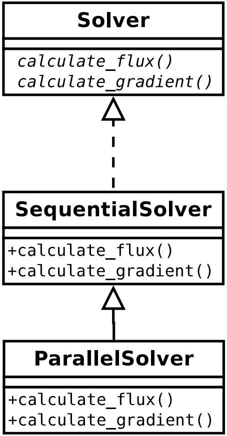
Two Types of Parallel Computing



- Simultaneously use shared-memory and distributed-memory parallel computing (DMPC & DMPC, respectively).
  - Main difference: Addressing space.
- Inter-process communication is needed.
  - DMPC is much more complex than SMPC.
  - DMPC determines the scalability.
  - MapReduce is unsuitable.

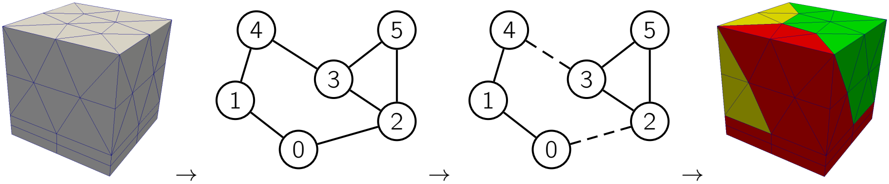
Extending Solver Kernel for SMPC

- A `Solver` class can be extended to use shared-memory parallel computing.
- Only the spatial loops are modified.
- Can use pthread, OpenMP, CUDA, OpenCL, etc.



Domain Decomposition for DMPC

- Before computation: Domain decomposition.
  - Use connectivity data to build the graph of cells.
  - Partition the graph by calling SCOTCH library.
  - Use the partitioned graph to decompose mesh data.

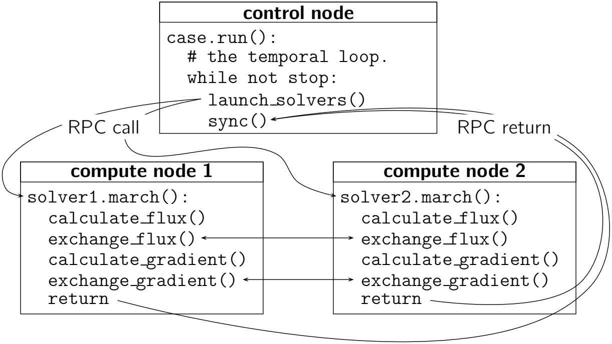


- During computation: Exchange data of the cells on the interface of different sub-domains.
  - Use MPI to communicate among sub-domains.

Solver Kernels Need Not Know DMPC

- DMPC is in SOLVCON framework.
- SMPC is in solver kernels.

DMPC Execution Flow in SOLVCON



- When developing solver kernels, we do not need to worry about the complexity of DMPC.
- Hybrid parallelism is achieved by the segregation.

Post-Processing is Bottleneck

- High-resolution simulations generate a lot of data:
    - For 50 million element mesh, the data for one scalar (single-precision) are 200 MB.
    - A typical run has at least 10,000 time steps.
    - Transient analysis: $10,000 \times 200\,\text{MB} = 2\,\text{TB}$.
    - $\rho, p, T, \vec{v}, \vec{\omega}$ for CFD: $2\,\text{TB} \times 9 = 18\,\text{TB}$.
- Workaround: Reducing output frequency.
    - Every 100 time steps: 180 GB.
- Post-processing the solutions is painfully time-consuming:
    - The large data are usually processed by using a single workstation.
    - Turnaround time could be in months.

Solutions in SOLVCON

- Parallel I/O.
  - Each sub-domain outputs its own solutions.
  - It is used with parallel post-processing.
- In situ visualization.
  - Visualization is being done on the fly with the simulation.
  - Everything happens in memory.
  - Output only graphic files, which are much smaller than the full solution field.
- Parallel I/O and in situ visualization are complementary to each other.

Python: Rich Ecosystem for Scientists

- Robust fundamental tools: NumPy, SciPy, Matplotlib, iPython, Cython.
- Abundant applications: Pandas, NLTK, networkx, VTK, PyMOL, Pyomo, ..., etc.
- iPython notebook is an excellent workbench from prototyping to presentation.
- Cython can help us to get the speed of C.
- A “virtual lab” can be built upon Python, like what SOLVCON is approaching.

Coding HPC for Research

- Identifying the fundamental structure.
  - In SOLVCON it's the two-loop structure.
  - Enabled by the insights from the "domain experts".
- Use Python from the beginning and to the end.
  - Prototype your system with Python.
  - Gradually replace performance hot spots with Cython or low-level C.
  - Try to stay away from C++ or Fortran as much as possible.
- It's very productive.
  - SOLVCON is multi-physics by its clear structure with hybrid parallelism.
  - Python rocks.

Thanks!