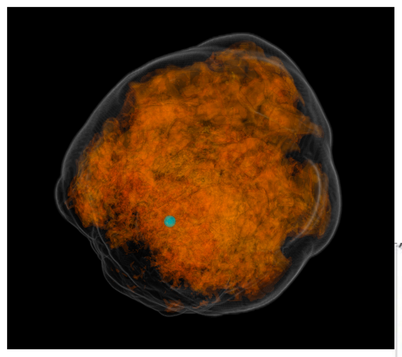
PERFORMANCE PORTABILITY FOR FLASH WITH TRANSPILATION

Saurabh Chawdhary[1], Mateusz Bysiek[2], Mohamed Wahib[2], Anshu Dubey[1]
[1]Mathematics and Computer Science Division, Argonne National Laboratory
[2]Tokyo Institute of Technology

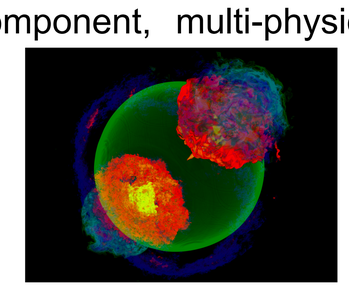
MOTIVATION

- Due to exponential rise of heterogeneity in computing, scientific codes like FLASH have a new challenge with performance portability.
- Lifecycle of scientific codes is several times that of platforms and devices.
- Writing code for every new device is difficult and time-consuming task, and can also lead to combinatorics explosion.
- Abstractions to enable architecture independence become necessary for sustainability of HPC scientific codes.
- FLASH5 combines software design with hierarchical composability at framework level, and code transformation at physics kernel level with transpilation to achieve ***performance portability***.
- The focus of TEAMS work is to apply code transformation on physics kernels in collaboration with Tokyo Institute of Technology

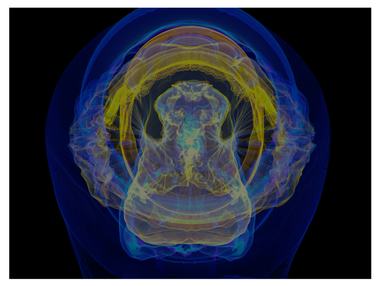
FLASH is a multi-component, multi-physics code serving several science domains.



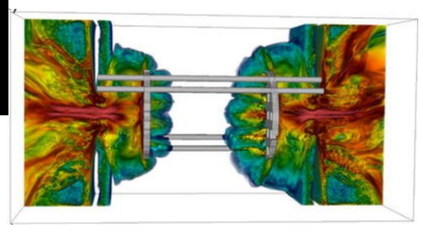
Core collapse supernova



Type Ia supernova



Galaxy clusters

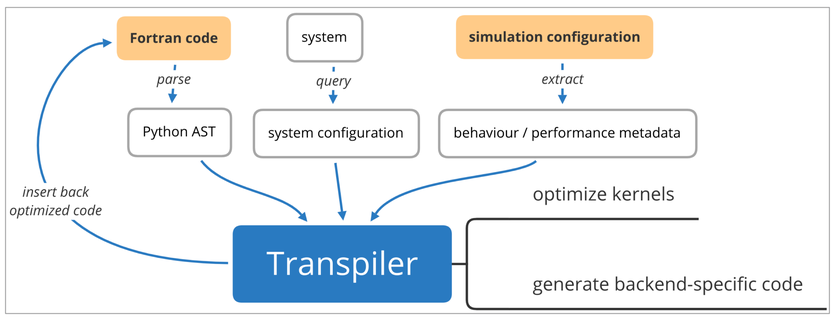


Laser experiments

Fig.1:.A sample pf FLASH simulations

METHOD

- Translation + Compilation = Transpilation.



- During the pre-compilation *setup phase*, the individual components of code are selected, dependencies sorted out and a curated application is generated for simulation at hand.

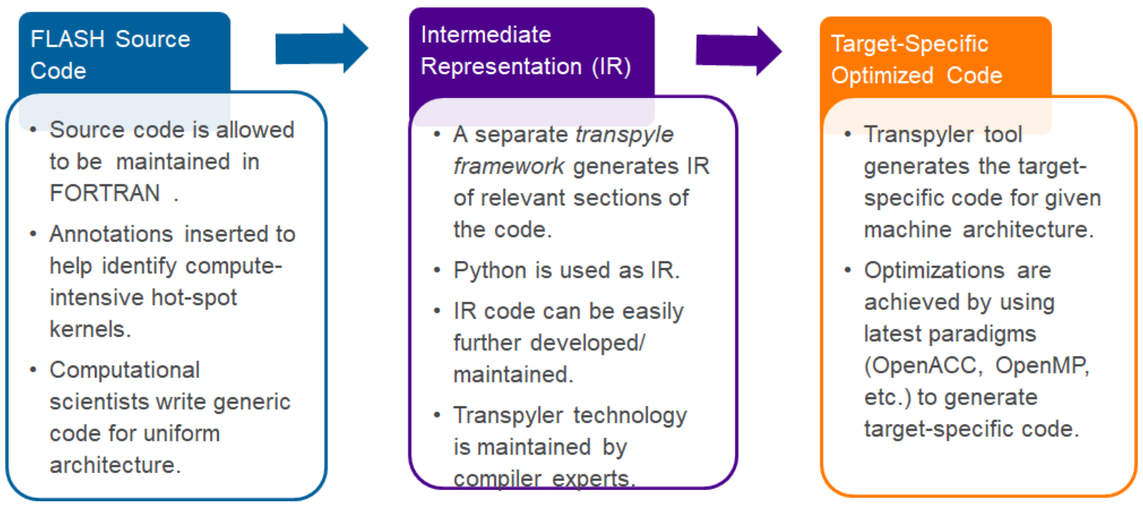


Fig. 2: Steps for realization of performance portability.

- access to all configuration (FLASH stores it as files);
- FLASH internal preprocessing is finished;
- allow user to make adjustments after setup (normally also allowed);
- allow user to re-compile after additional hand-tuning of the transpiled code.

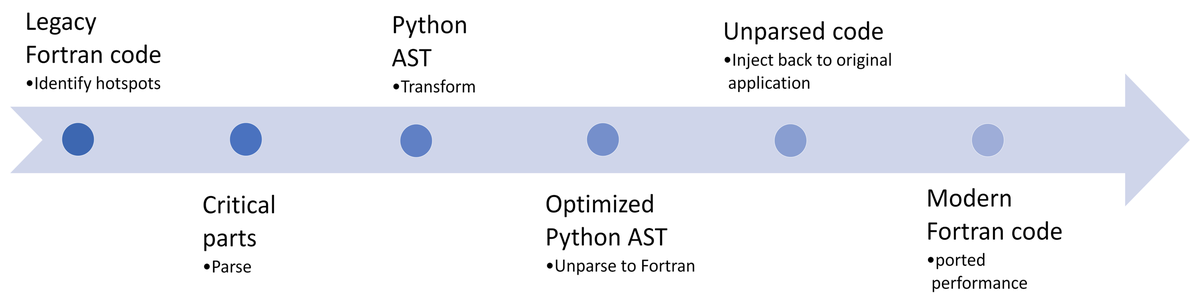


Fig. 3: FLASH optimization using *transpyle* framework.

- The *transpyler* automatically alters the code towards most efficient execution depending on:
  - internal structure of the kernels (operations, data dependencies, locality);
  - relationships between different kernels;
  - system architecture (is it multi-core, many-core, GPU, or heterogeneous);
  - simulation configuration (blocks per rank, grid size, etc.).

AT FLASH END

- Convert code to fine-grained kernels – better exposure for optimization possibilities
  - Transpiler inlines functions – avoid overhead of function calls
  - Code still in Fortran
- Identify hot-spot kernels for experimentation

CURRENT STATE

- Pikced hot-spot kernels using profiling tools (score-P and hpctoolkit).
- Code transformation with transpyler is verified using Sod shock tube, and supernova simulations.
- Transpyler is able to generate target code for CPU and GPU

FUTURE WORK

- Improvements in the performance of transpyler generated code.

ACKNOWLEDGEMENTS: This work was partially funded by OASCR under the SciDAC TEAMS project,

[1] Transpyle framework *:* https://github.com/mbdevpl/transpyle

[2] FLASH 5 alpha release : https://github.com/ECP-Astro/FLASH5