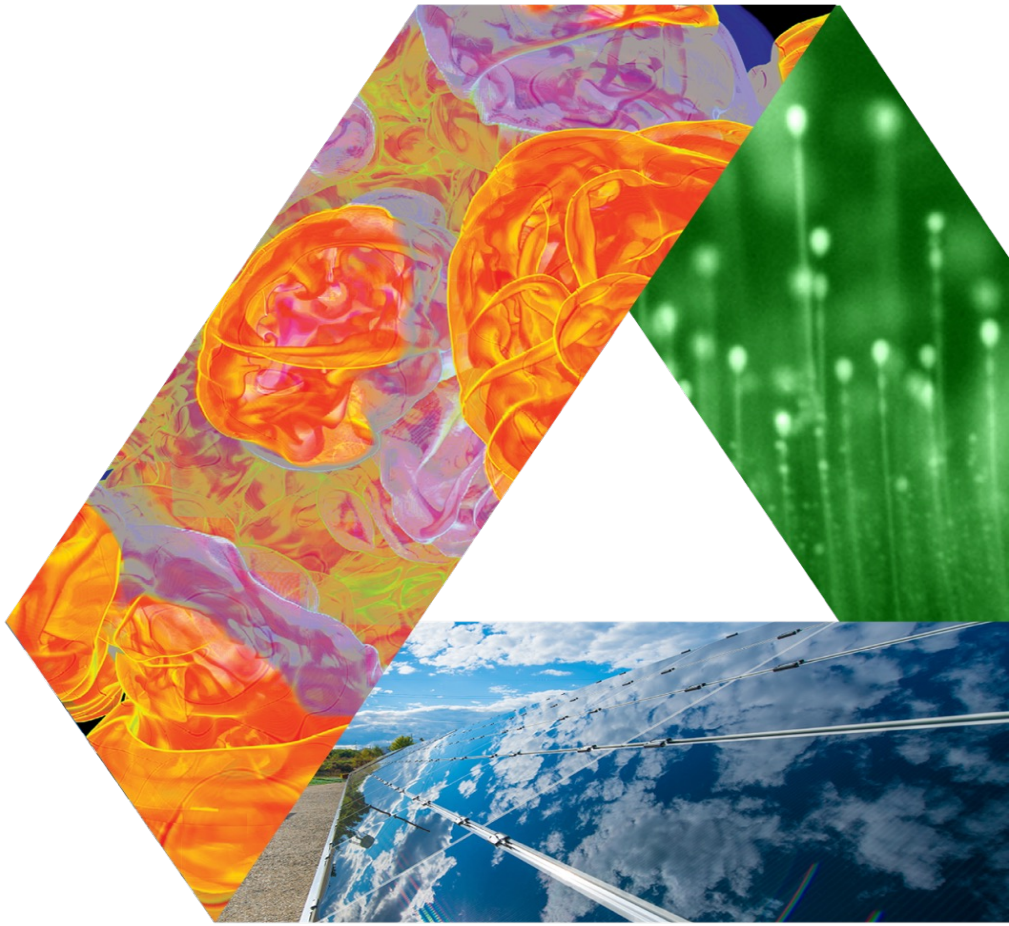
IDEAS productivity

Making Codes Performance Portable Without Using C++ Abstraction

Anshu Dubey

Advanced Modeling & Simulation (AMS) Seminar Series

NASA Ames Research Center, April 25th, 2024



Argonne National Laboratory

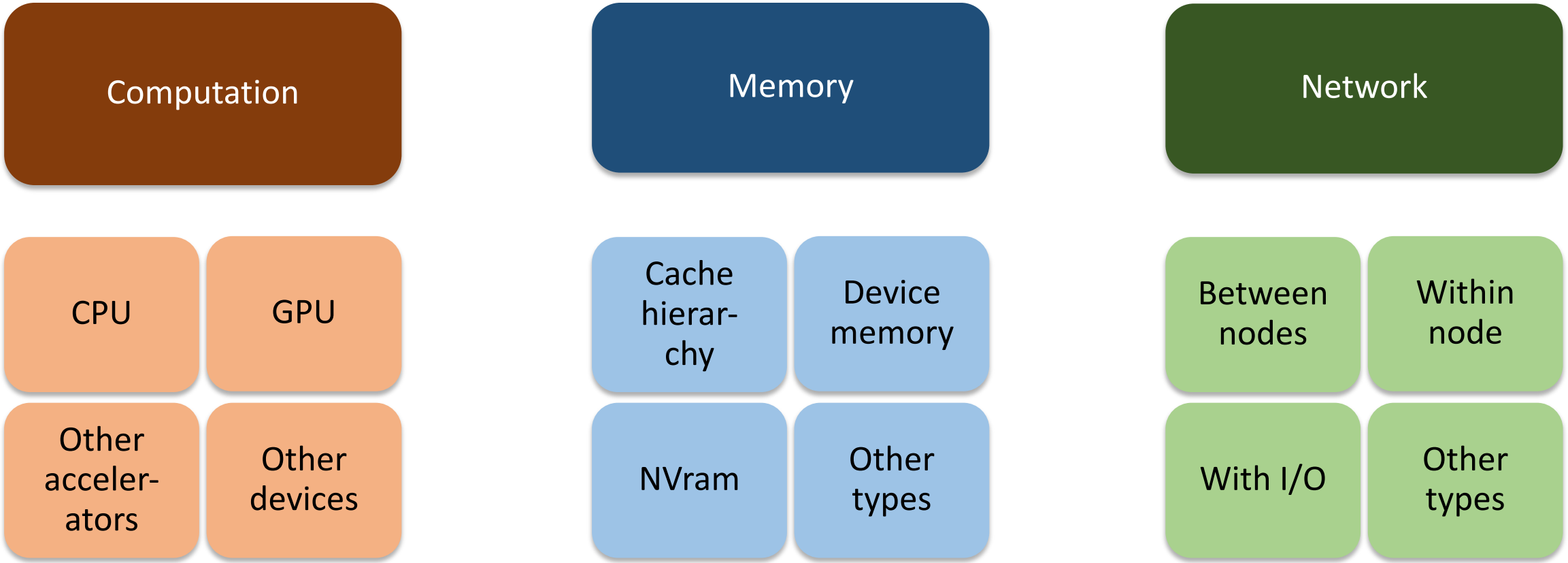
ECP EXASCALE COMPUTING PROJECT

U.S. DEPARTMENT OF ENERGY | Office of Science

acknowledgements

- This work was partly supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
- Flash-X authors -- A. Dubey, K. Weide, A. Dhruv, S. Couch, J.A. Harris, J. O'Neal, Y. Lee, T. Klosterman, W, Kwiecinski, R. Jain, O.E.B. Messer, M. Pajkos, J. Carlson, R. Chu, S. Chawdhary, P.M. Ricker, D. Lee, K. Antypas, K.M. Riley, C. Daley, M. Ganapathy, F.X. Timmes, D.M. Townsley, M. Vanella, J. Bachan, P.Rich, S. Kumar, E. Endeve, W. R. Hix, A. Mezzacappa, T. Papatheodore

Platform Heterogeneity



Mechanisms Needed by the Code

Mechanisms to unify expression of computation

- Minimize maintained variants of source suitable for all computational devices
- Reconcile differences in data structures

Mechanisms to move work and data to computational targets

- Moving between devices
  - Launching work at the destination
  - Hiding latency of movement
- Moving data offnode

Mechanisms to map work to computational targets

- Figuring out the map
  - Expression of dependencies
  - Cost models
- Expressing the map

Mechanisms Needed by the Code

Mechanisms to unify expression of computation

- Minimize maintained variants of source suitable for all computational devices
- Reconcile differences in data structures

Mechanisms to move work and data to computational targets

- Moving between devices
  - Launching work at the destination
  - Hiding latency of movement
- Moving data offnode

Mechanisms to map work to computational targets

- Figuring out the map
  - Expression of dependencies
  - Cost models
- Expressing the map

So what do we need?

- Abstractions layers
- Code transformation tools
- Data movement orchestrators

Mechanisms Needed by the Code

Mechanisms to unify expression of comput
- Minim … ource suitab … ices
- Reconcile differences in data structures

Macroprocessor

Mechanisms to move work and data to comp…
- Movin…
  - La… ation
  - Hiding latency of movement
- Moving data offnode

Milhoja

Mechanisms to map work to comput…
- Figurin…
  - Ex…
  - Cost models
- Expressing the map

CG-Kit

So what do we need?

- Abstractions layers
- Code transformation tools
- Data movement orchestrators

Mechanisms Needed by the Code

Mechanisms to unify expression of computation

- Minimize maintained variants of source suitable for all computational devices
- Reconcile differences in data structures

Express code with embedded macros

- Let macros have multiple alternative definitions
- Implement mechanism to select specific macro definition
- Implement mechanism to safely include more than one definition
- Allow inline, recursion and arguments in macros

Mechanisms Needed by the Code

Version 1

```
integer,parameter:: N=10, M=12
integer:: i,j
real,dimension(N,M) :: A,B
real,dimension(N) :: temp

do j=1,n
   do i=1,m
      temp(i)=A(i,j)*0.5
      A(i,j) = A(i,j)*B(i,j)
   end do
   do i=1,m
      B(i,j)=temp(i)+A(i,j)
   end do
end do
```

Version 2

```
integer,parameter:: N=10, M=12
integer:: i,j
real,dimension(N,M) :: A,B
real :: temp


do j=1,n
   do i=1,m
      temp=A(i,j)*0.5
      A(i,j) = A(i,j)*B(i,j)
      B(i,j)=temp+A(i,j)
   end do
end do
```

Unified Code

```
integer,parameter:: N=10, M=12
  integer:: i,j
  real,dimension(N,M) :: A,B
  @M declare

   do j=1,n
      do i=1,m
         @M tmp =A(i,j)*0.5
         A(i,j) = A(i,j)*B(i,j)
      @M break_loop
         B(i,j) = @M tmp +A(i,j)
      end do
 end do
```

Mechanisms Needed by the Code

Unified Code

```
integer,parameter:: N=10, M=12
  integer:: i,j
  real,dimension(N,M) :: A,B
  @M declare

   do j=1,n
     do i=1,m
        @M tmp =A(i,j)*0.5
        A(i,j) = A(i,j)*B(i,j)
     @M break_loop
        B(i,j) = @M tmp +A(i,j)
     end do
end do
```

```
[declare]
definition =
  real,dimension(N) :: temp

[tmp]
definition =
  temp(i)

[break_loop]
definition =
  end do
  do i=1,m
```

Version 1

```
  integer,parameter:: N=10, M=12
  integer:: i,j
  real,dimension(N,M) :: A,B
  real,dimension(N) :: temp

  do j=1,n
     do i=1,m
        temp(i)=A(i,j)*0.5
        A(i,j) = A(i,j)*B(i,j)
     end do
     do i=1,m
        B(i,j)=temp(i)+A(i,j)
     end do
end do
```

Mechanisms Needed by the Code

Unified Code

```
integer,parameter:: N=10, M=12
  integer:: i,j
  real,dimension(N,M) :: A,B
  @M declare

   do j=1,n
     do i=1,m
        @M tmp =A(i,j)*0.5
        A(i,j) = A(i,j)*B(i,j)
     @M break_loop
        B(i,j) = @M tmp +A(i,j)
     end do
end do
```

```
[declare]
definition =
  real :: temp

[tmp]
definition =
  temp

[break_loop]
definition =
```

Version 2

```
  integer,parameter:: N=10, M=12
  integer:: i,j
  real,dimension(N,M) :: A,B
  real :: temp


  do j=1,n
     do i=1,m
        temp=A(i,j)*0.5
        A(i,j) = A(i,j)*B(i,j)
        B(i,j)=temp+A(i,j)
     end do
end do
```

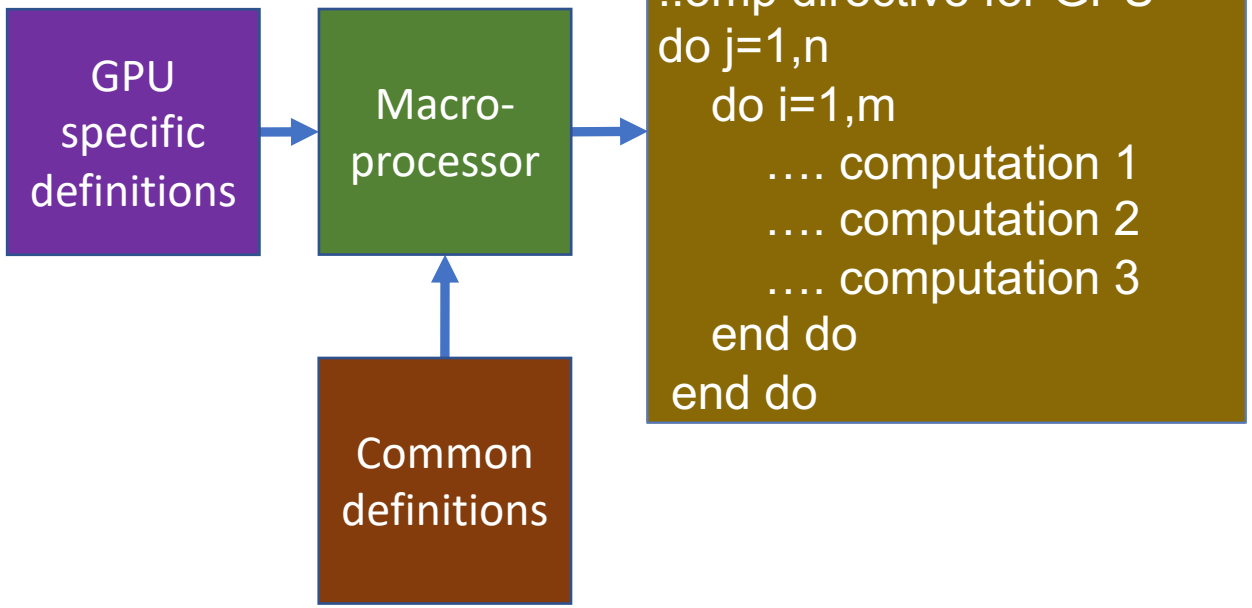
```
@M declare
common declarations
@M directive1
@M loop_2d
      …. computation 1
      …. computation 2
@M endloop_2d_spl
@M directive2
@M loop_2d_spl
      …. computation 3
@M endloop_2d
```

Common definitions

```
[loop_2d]              [endloop_2d]
  definition=            definition=
  do j=1,n                 end do
   do i=1,m              end do
```

Definitions for GPU

```
[declare]
   definition=
  gpu-specific declarations
[directive1]
  definition=
  !!omp directive for gpu
[endloop_2d_spl]
 definition=
[directive2]
 definition=
[loop_2d_spl]
 definition=
```



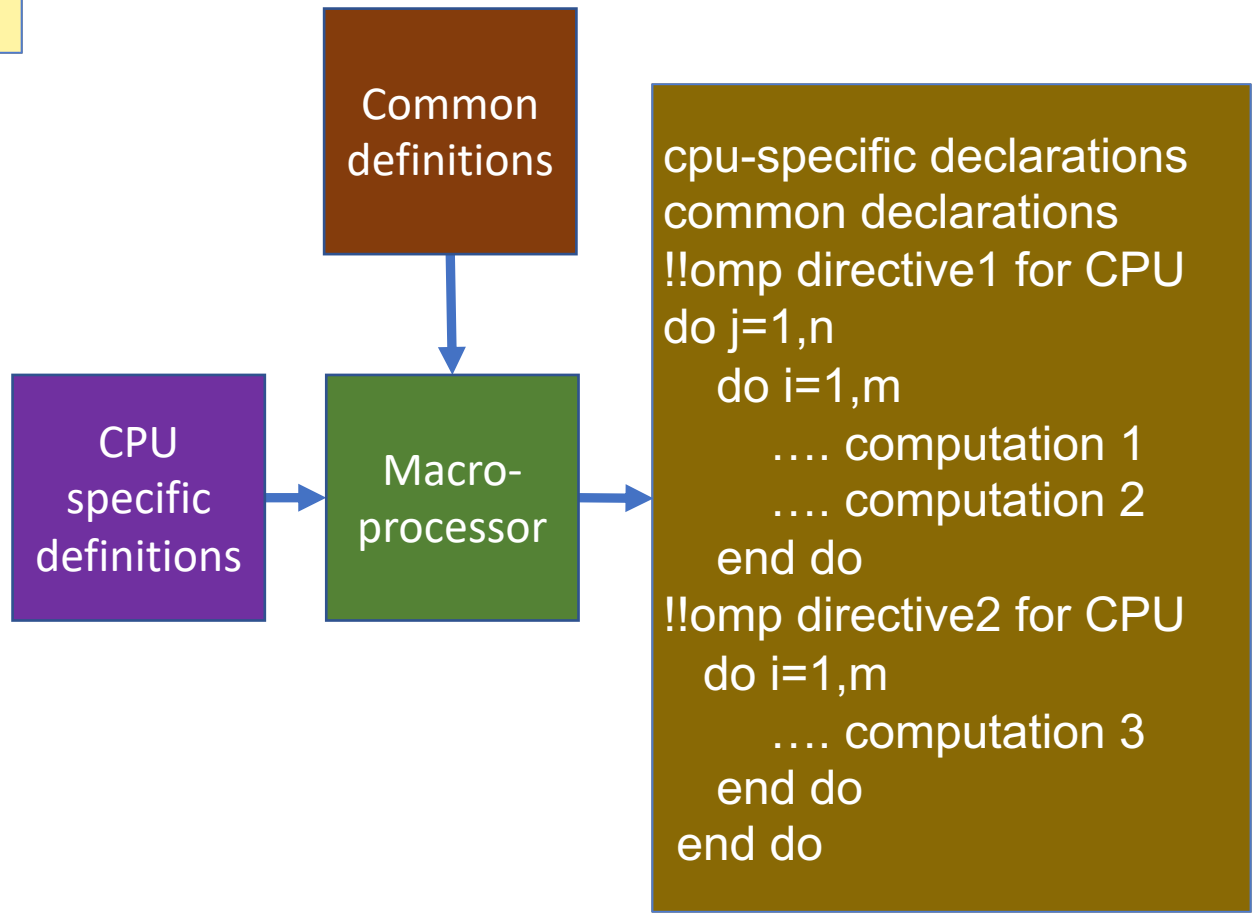
```
@M declare
common declarations
@M directive1
@M loop_2d
      …. computation 1
      …. computation 2
@M endloop_2d_spl
@M directive2
@M loop_2d_spl
      …. computation 3
@M endloop_2d
```

Common definitions

```
[loop_2d]            [endloop_2d]
  definition=          definition=
  do j=1,n               end do
   do i=1,m            end do
```

Definitions for CPU

```
[declare]
 definition=
   cpu-specific declarations
[directive1]
 definition=
  !!omp directive1 for cpu
[endloop_2d_spl]
 definition=
    @M endloop_2d
[directive2]
 definition=
!!omp directive2 for cpu
[loop_2d_spl]
 definition=
    @M loop_2d
```



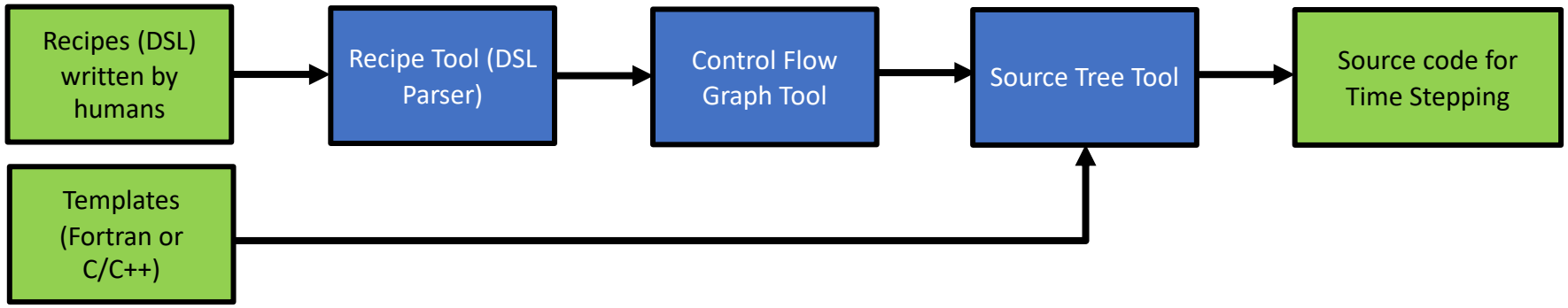
Mechanisms Needed by the Code

Mechanisms to map work to computational targets

- Figuring out the map
  - Expression of dependencies
  - Cost models
- Expressing the map

CGKit : Generating Code from Recipes and code Templates

CG-Kit ❑ Generating Code from Recipes and code Templates



Orthogonal separation of concerns

express dependencies

express hardware mapping

Example recipe

```
dIn = ConcurrentDataBegin()()

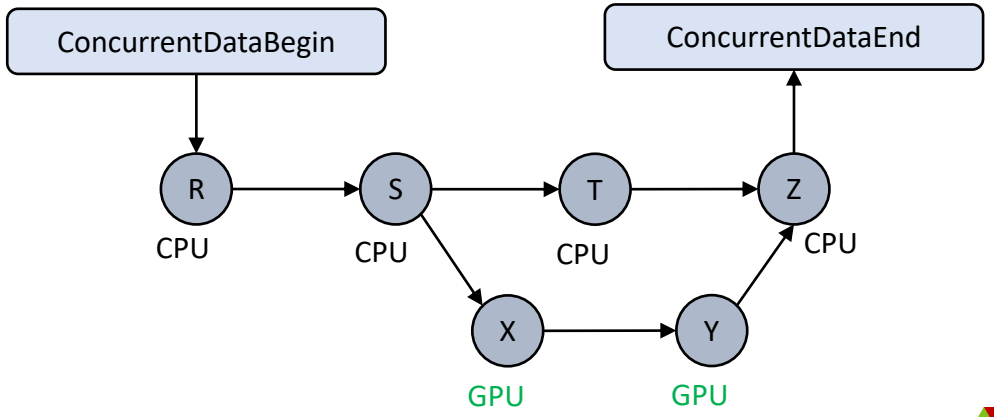
aR = Action(routine = 'function_R')(dIn)
aS = Action(routine = 'function_S')(aR)
aT = Action(routine = 'function_T')(aS)

aX = Action(routine = 'function_X')(aS)
aY = Action(routine = 'function_Y')(aX)
aZ = Action(routine = 'function_Z')([aT,aY])

dOut = ConcurrentDataEnd()(aZ)

ConcurrentHardware(
    CPU={'actions': [aR,aS,aT,aZ]},
    GPU={'actions': [aX,aY]}
)
```

Resulting control flow graph



ECP EXASCALE COMPUTING PROJECT
Argonne NATIONAL LABORATORY

Mechanisms Needed by the Code

Mechanisms to move work and data to computational targets

- Moving between devices
  - Launching work at the destination
  - Hiding latency of movement
- Moving data offnode

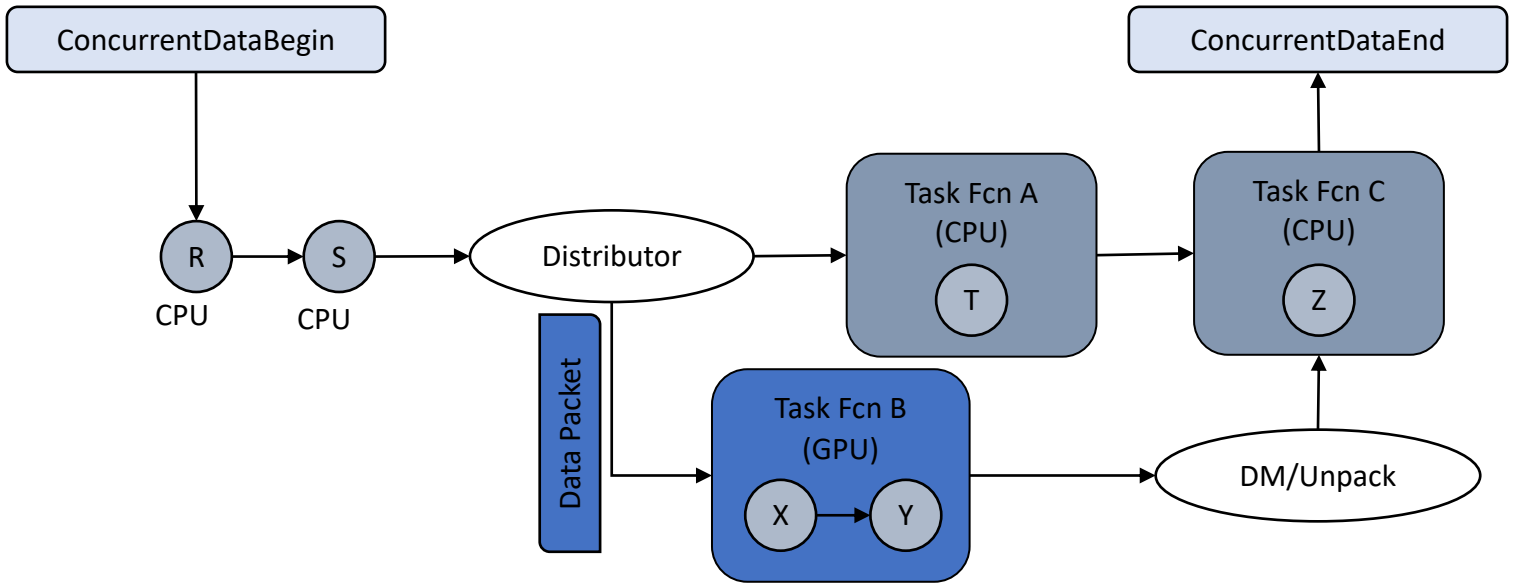
Data type

Milhoja: An extended finite state machine to move data and computation

Milhoja

- A mechanism to move data and computation between devices

Control flow graph (With Milhoja)

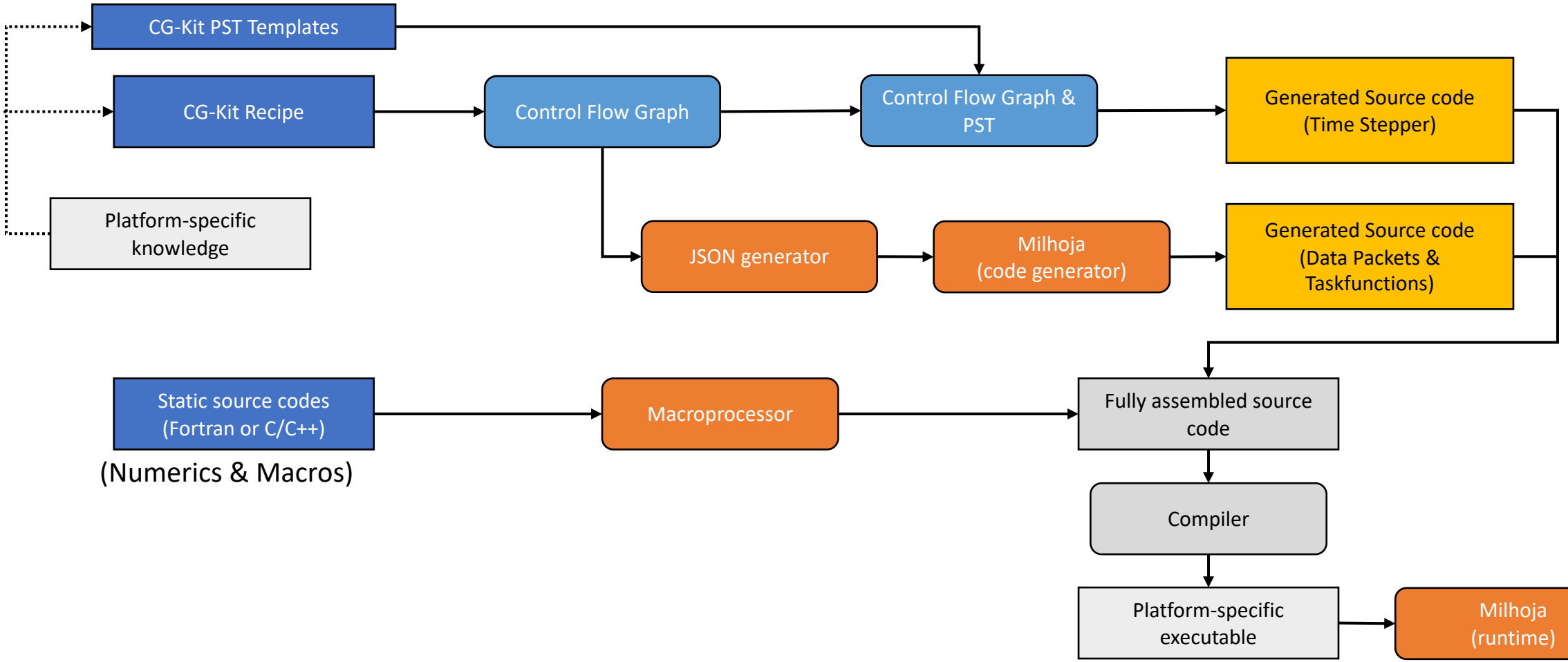


Code Generators

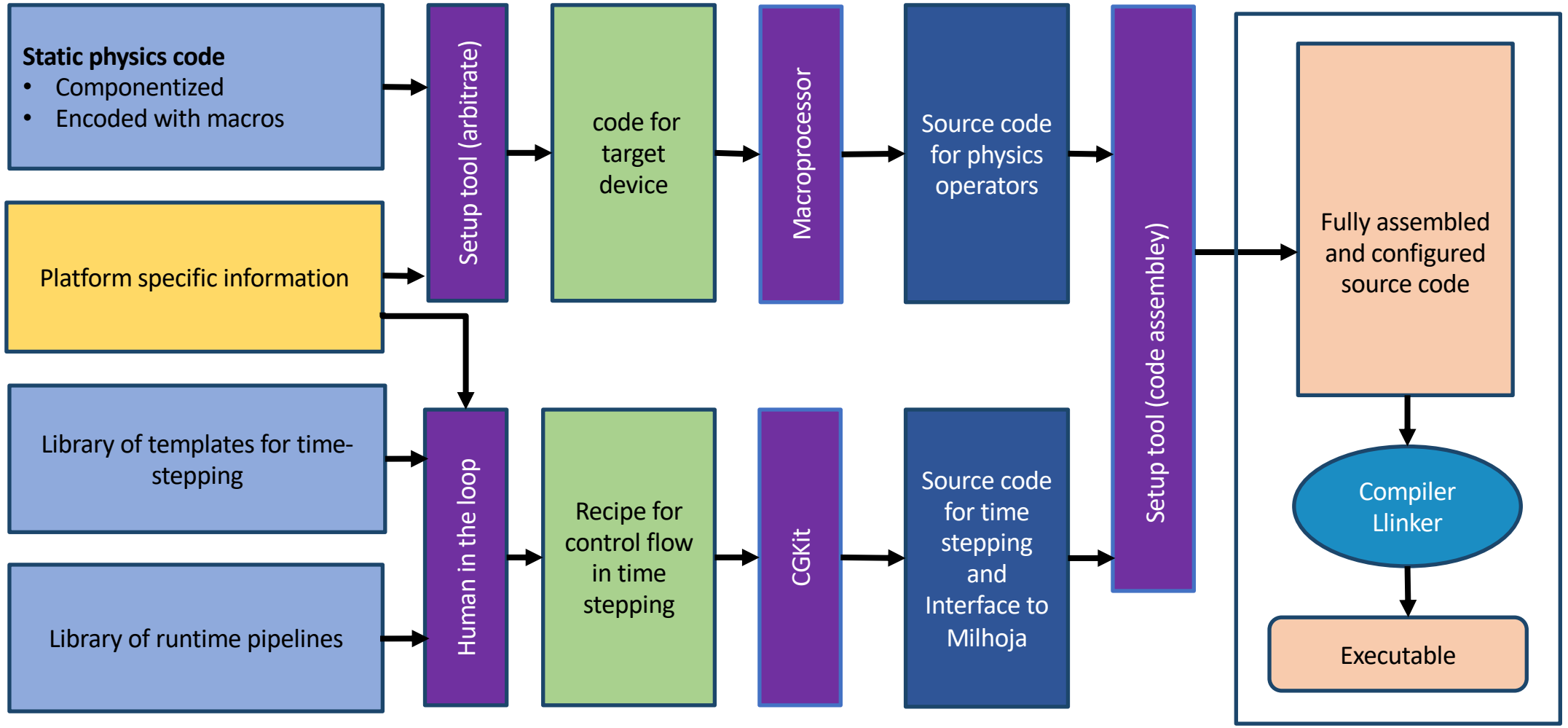
- Two Classes
  - Data packet generators
    - Parse the interface files
    - Collect all data to be put into a data packet
    - Generate code that will flatten all data into data packets
  - Task function generators
    - Consolidate functions to be invoked
    - Bookended by internode communication
    - Unpack data packets
- Decorate interface definitions with needed metadata

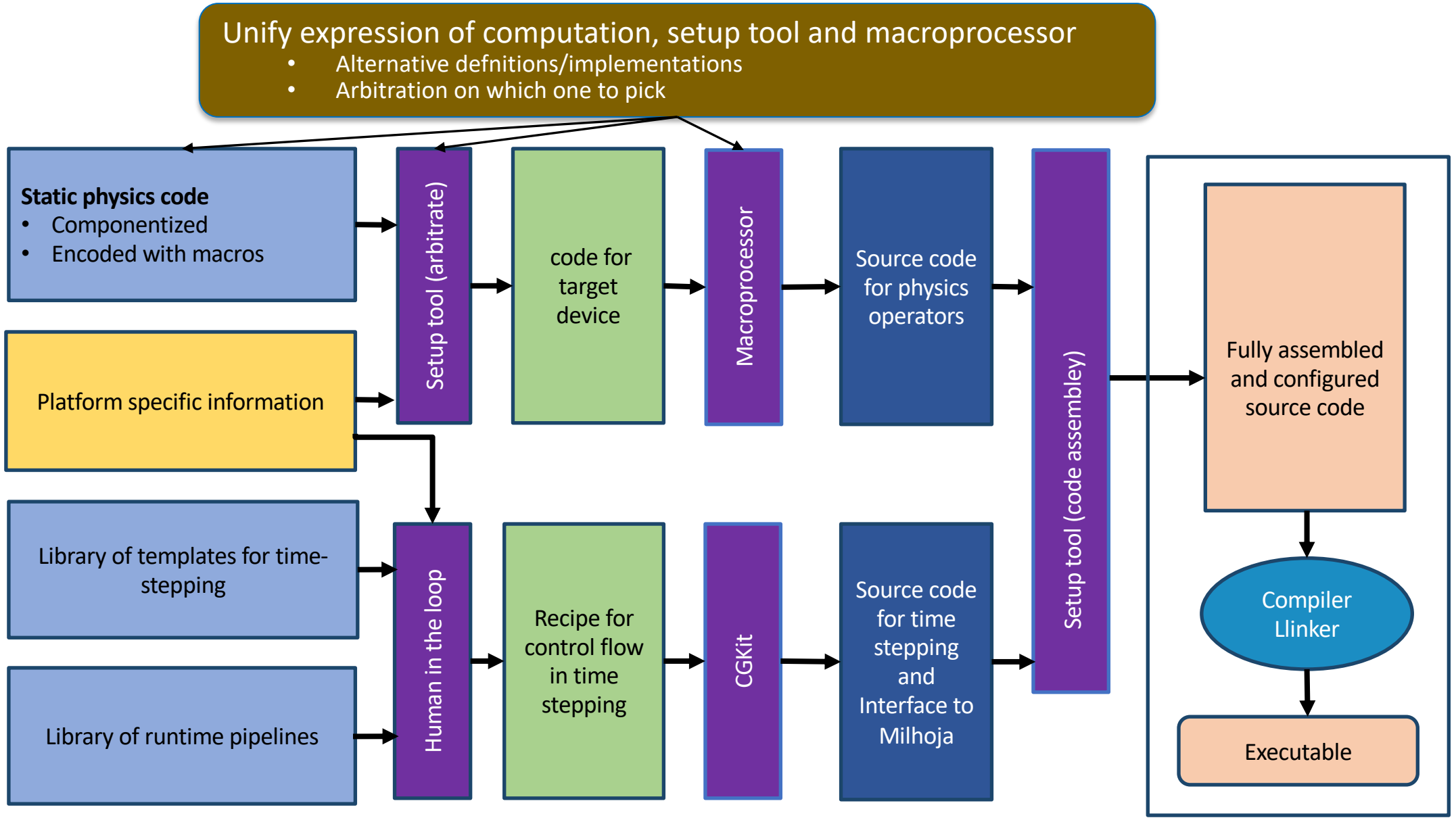
  Example

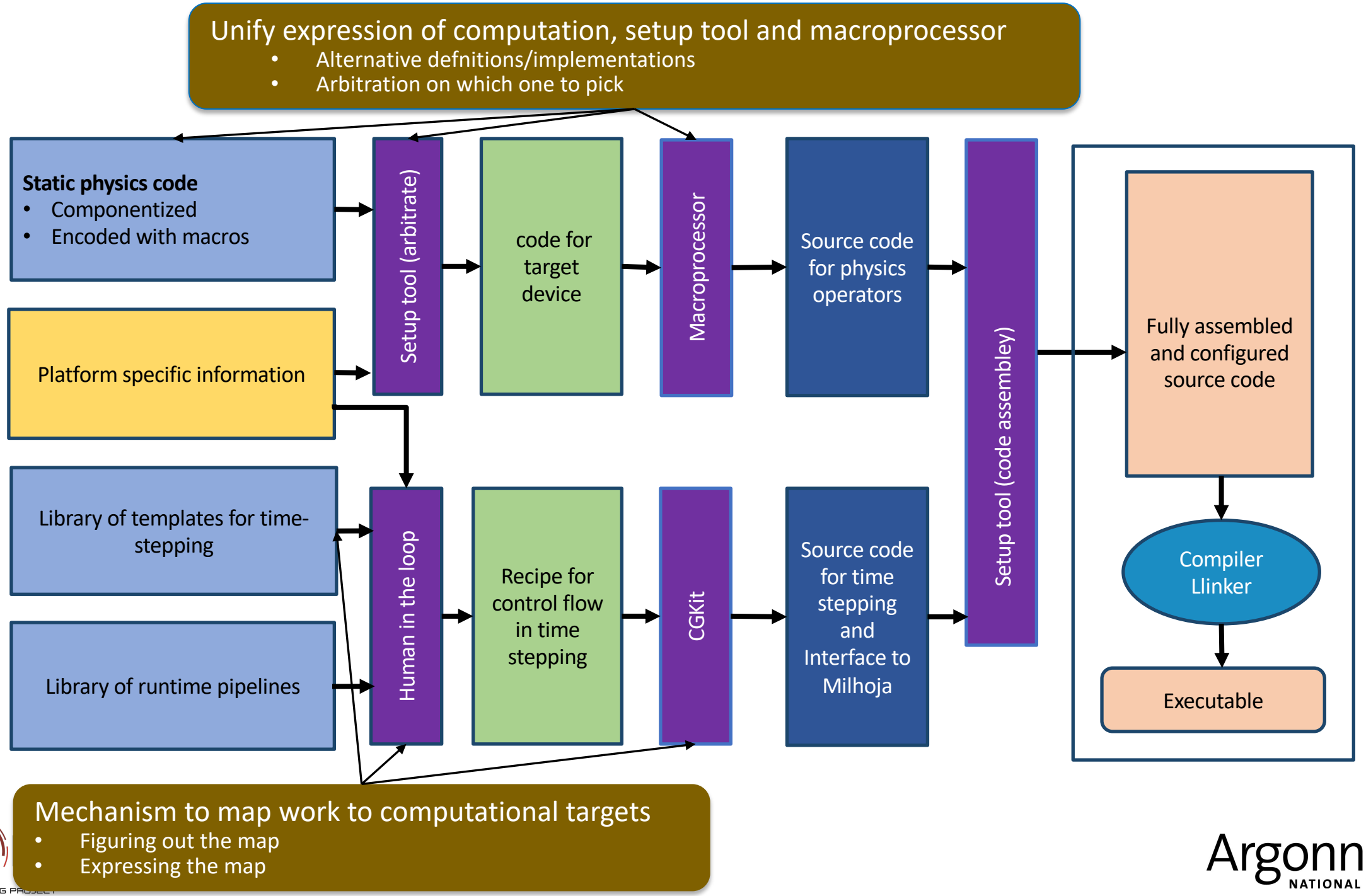
Code generation



Code Assembly

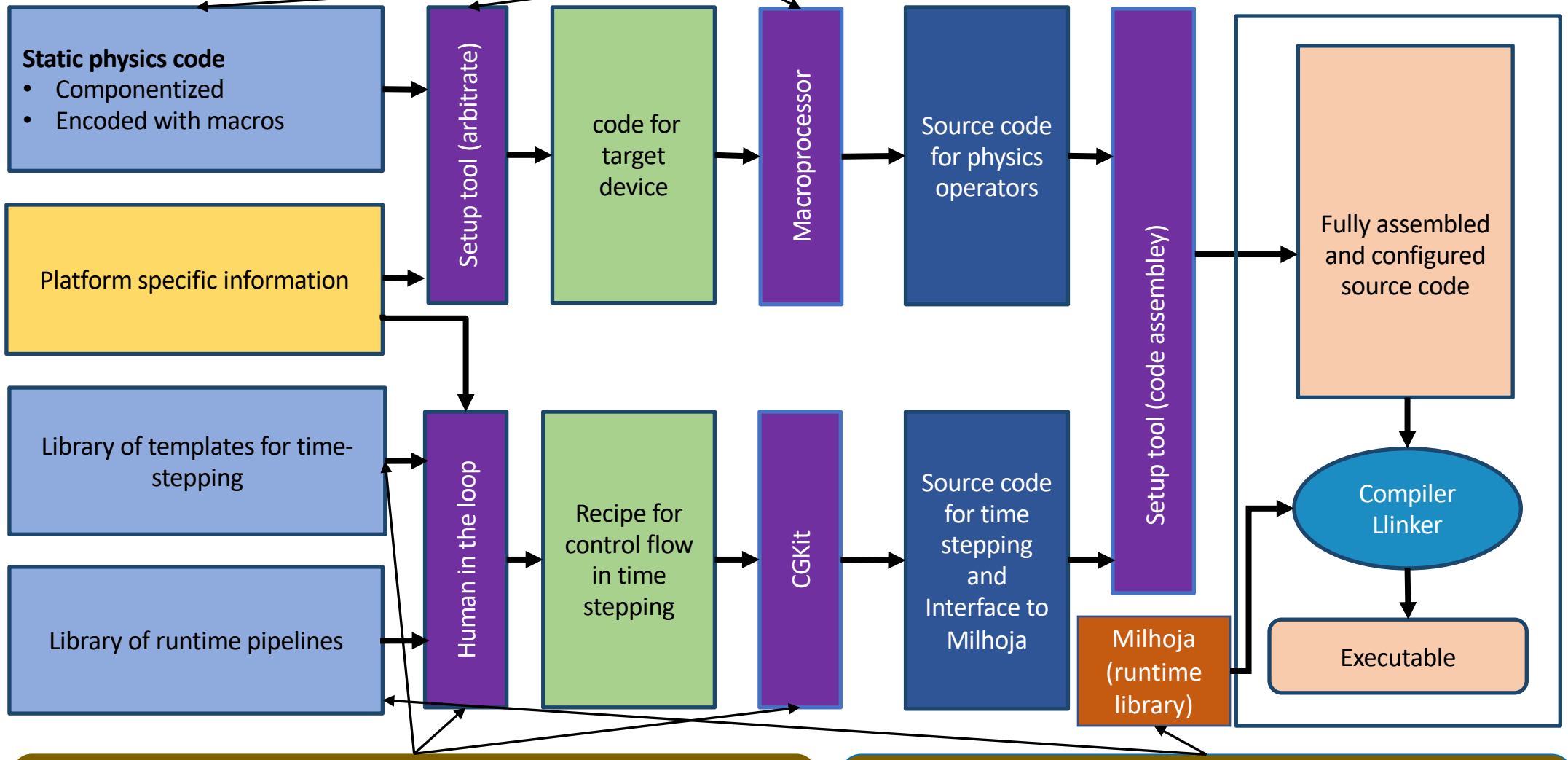



Unify expression of computation, setup tool and macroprocessor
• Alternative defnitions/implementations
• Arbitration on which one to pick
Static physics code
• Componentized
• Encoded with macros
Platform specific information
Library of templates for time-stepping
Library of runtime pipelines
Setup tool (arbitrate)
Human in the loop
code for target device
Recipe for control flow in time stepping
Macroprocessor
CGKit
Source code for physics operators
Source code for time stepping and Interface to Milhoja
Setup tool (code assembley)
Fully assembled and configured source code
Compiler Llinker
Executable


Unify expression of computation, setup tool and macroprocessor
• Alternative defnitions/implementations
• Arbitration on which one to pick
Static physics code
• Componentized
• Encoded with macros
Platform specific information
Library of templates for time-stepping
Library of runtime pipelines
Setup tool (arbitrate)
Human in the loop
code for target device
Recipe for control flow in time stepping
Macroprocessor
CGKit
Source code for physics operators
Source code for time stepping and Interface to Milhoja
Setup tool (code assembley)
Fully assembled and configured source code
Compiler Llinker
Executable
Mechanism to map work to computational targets
• Figuring out the map
• Expressing the map

Unify expression of computation, setup tool and macroprocessor

- Alternative definitions/implementations
- Arbitration on which one to pick



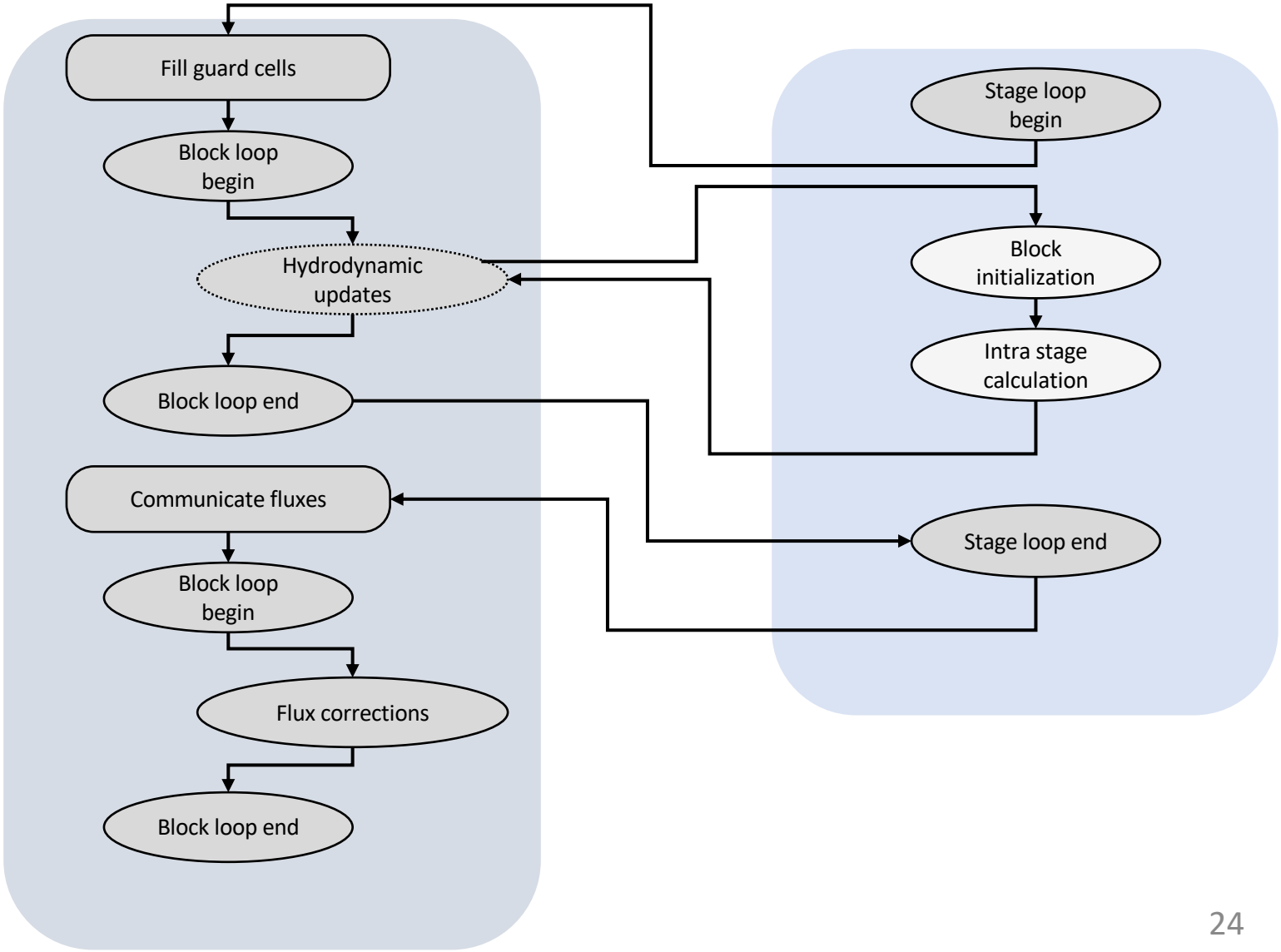
Mechanism to map work to computational targets

- Figuring out the map
- Expressing the map

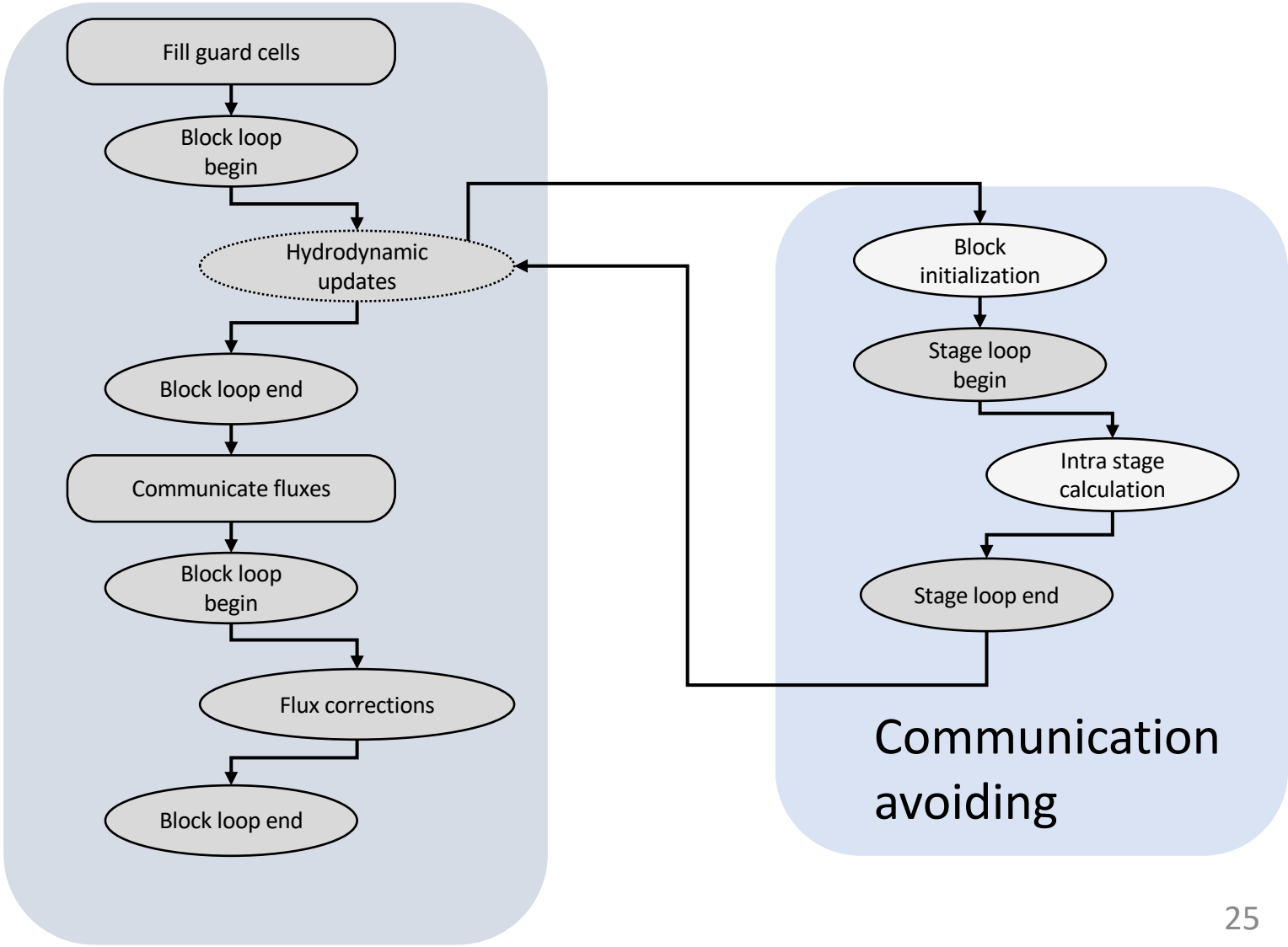
Mechanism to move work and data to targets

- Moving between devices
- Hiding latency of movement

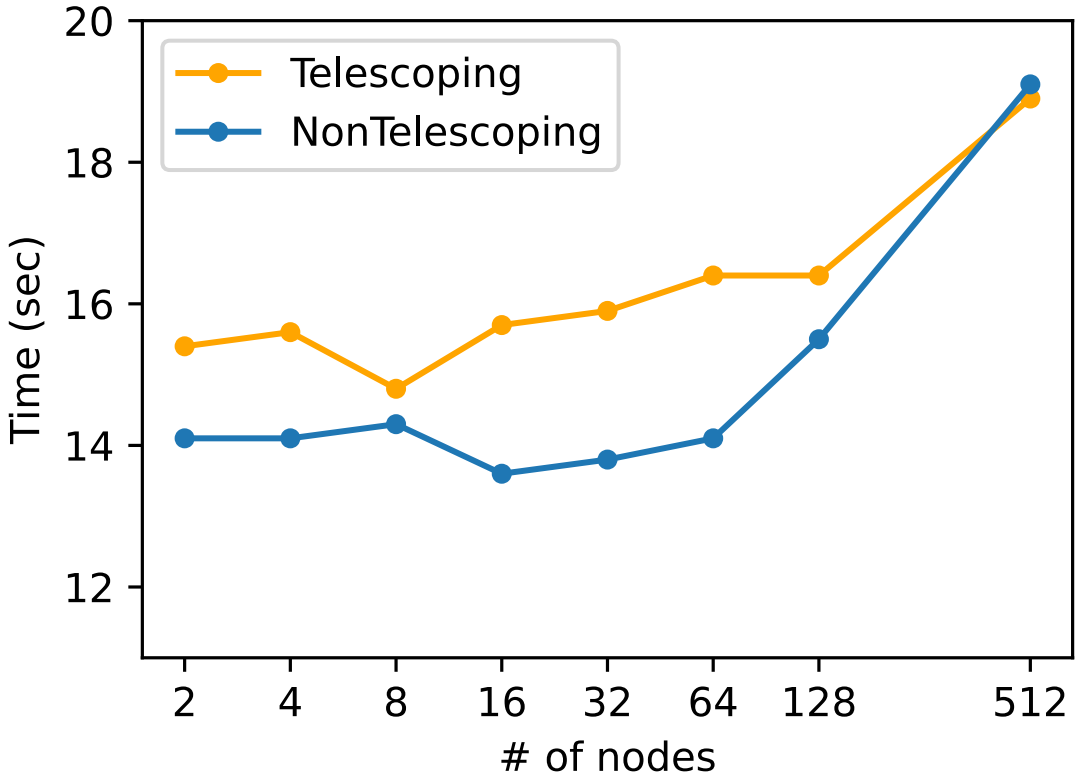
Example: Two Hydro Variants



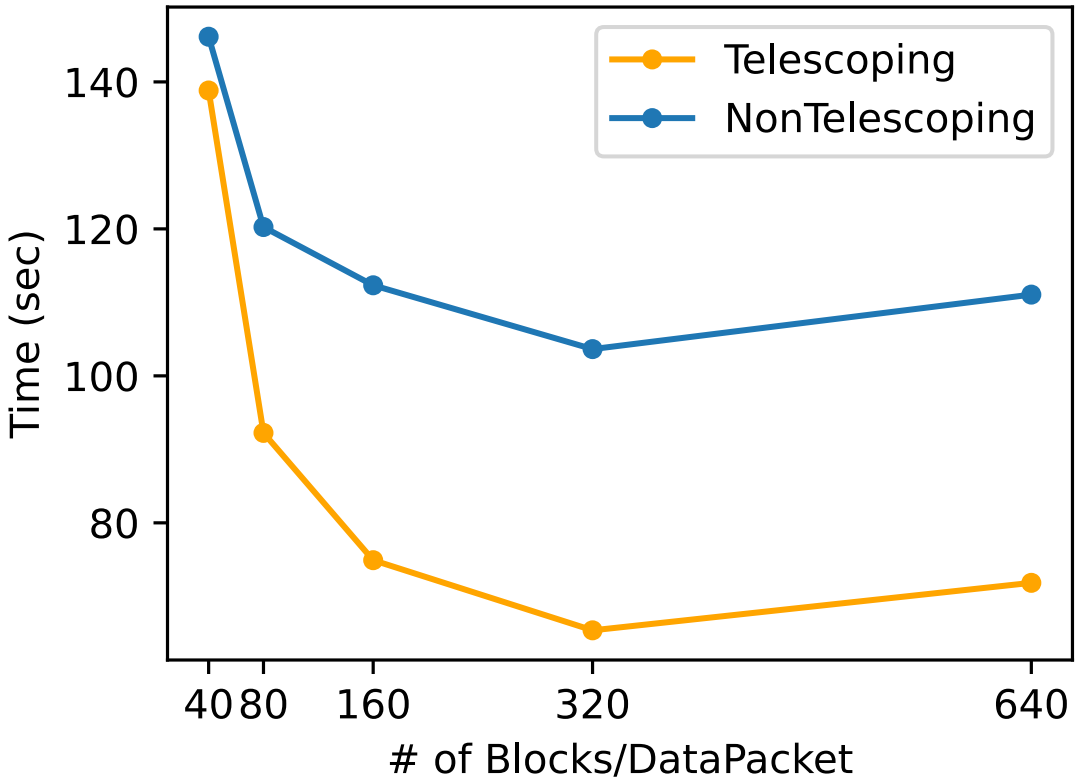
Example: Two Hydro Variants



Advantages of Orchestrating

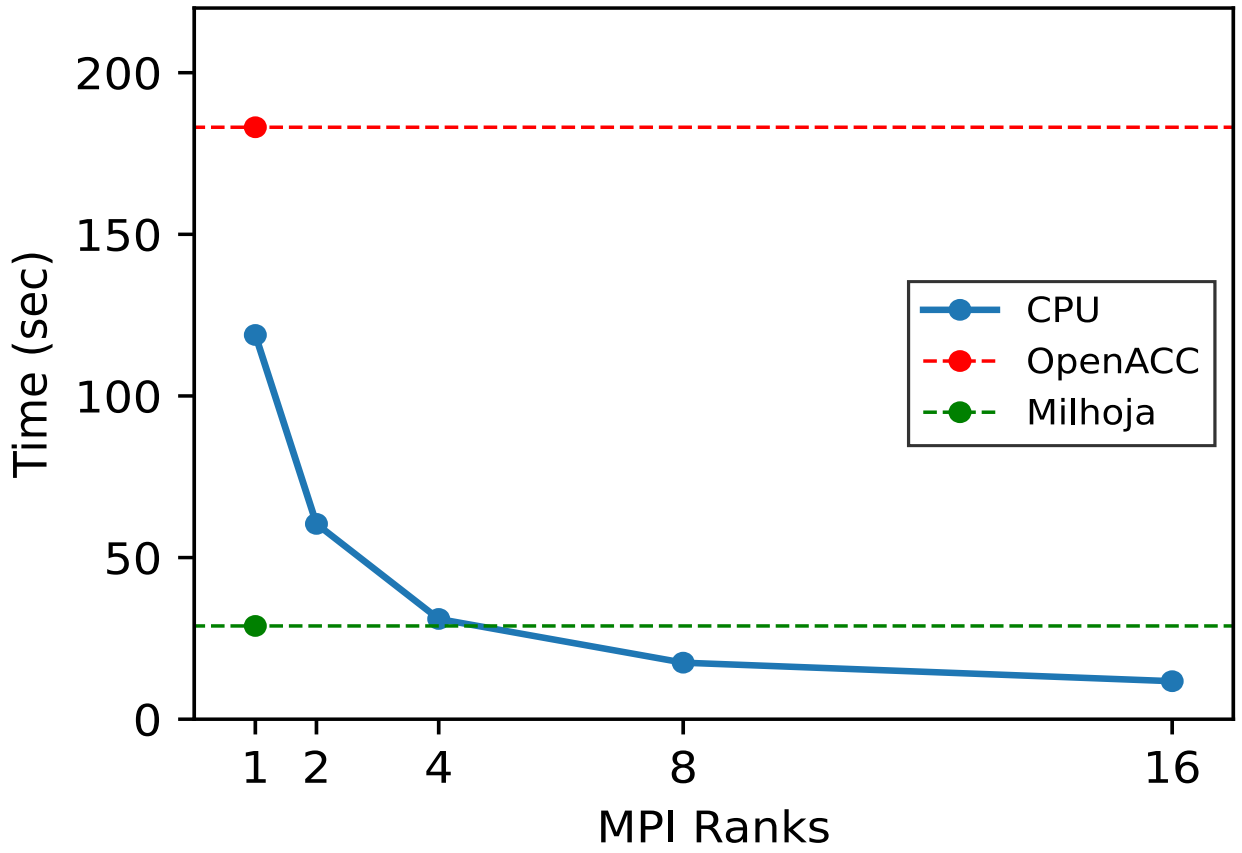


Performance comparisons on the CPU-only machine (Fugaku)



Performance comparisons with GPU (Nvidia RTX 3080)

Advantage of Orchestrating



Offloading performance of Hydro

ECP EXASCALE COMPUTING PROJECT
Argonne NATIONAL LABORATORY

Porting to a new platform

- In an ideal world
  - Add to the library of runtime pipelines
  - Add to the library of recipes templates
  - Add to the knowledge base of the performance model
- In real world
  - Add variants for some solvers with alternative definitions of keys
- In the worst case
  - Develop new algorithms and add whole alternative implementation for some solvers

References

- Dubey, Anshu, et al. "**Flash-X: A multiphysics simulation software instrument.**" *SoftwareX* 19 (2022): 101168.
- Rudi, Johann, et al. "**CG-Kit: Code Generation Toolkit for Performant and Maintainable Variants of Source Code Applied to Flash-X Hydrodynamics Simulations.**" *arXiv preprint arXiv:2401.03378* (2024).
- O'Neal, Jared, et al. "**Domain-specific runtime to orchestrate computation on heterogeneous platforms.**" *European Conference on Parallel Processing*. Cham: Springer International Publishing, 2021.
- Dubey, Anshu, et al. "**A tool and a methodology to use macros for abstracting variations in code for different computational demands.**" *Future Generation Computer Systems* (2023).

Questions .....

Spark Variants (GRID)

- Flash-X supports two block-structured AMR grid backends
  - Paramesh: Octree-based
  - AMReX: Level-based
- Each has different preferences for flux correction at fine-coarse boundaries

```
do all_blocks
  ! hydrodynamics updates
end do
call communicate_fluxes() ! p2p communication
do all_blocks
  ! flux correction
end do
```

Paramesh

```
do lev = max_level, 1, -1
  call communicate_fluxes() ! p2p communication
  do blocks_on(level = lev)
    ! hydrodynamics updates
    ! flux correction
  end do
end do
```

AMReX

Spark Variants (RK Modes)

- Spark adopts SSP-RK methods for integrating the solution
  - Requires halo-exchanges in each RK sub-stage
    - It may not be performant on heterogeneous systems
- To avoid additional data communications, we introduced “Telescoping” mode
  - One halo-exchange in one full time-step, with extra layers of guard cells

```
do stage = 1, max_stage
  call fill_guardcells() ! p2p communication
  do all_blocks
    ! block initializations
    ! intra stage calculations
  end do
end do
```

Non-telescoping (traditional)

```
call fill_guardcells() ! p2p communication
do all_blocks
  ! block initializations
  do stage = 1, max_stage
    ! intra stage calculations
  end do
end do
```

Telescoping