A DSL for Performance Orchestration

Thiago Teixeira
David Padua
William Gropp

Department of Computer Science
University of Illinois at Urbana-Champaign

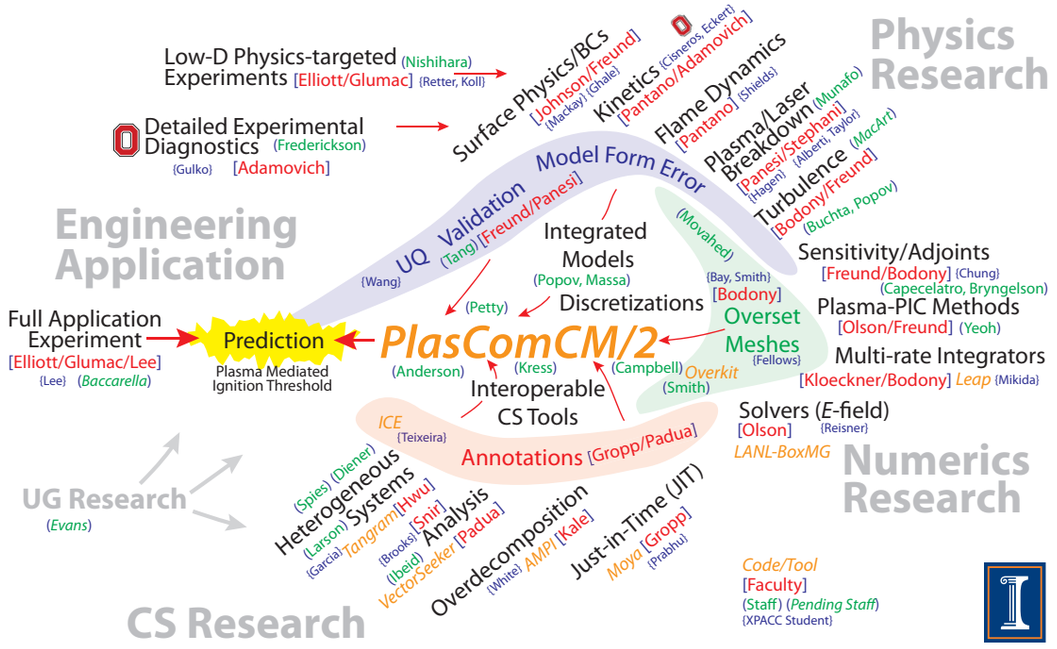
Scalable Tools Workshop
July, 2018

DOE/NNSA/ASC/PSAAPII:
The Center for Exascale Simulation of
Plasma-coupled Combustion

XPACC

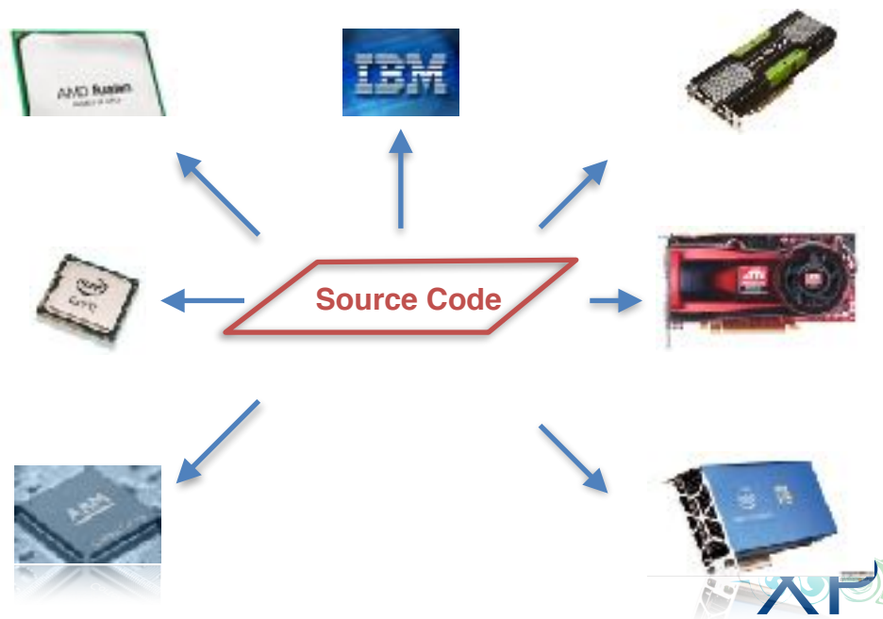
XPACC

- The Center for Exascale Simulation of Plasma-Coupled Combustion
- Developing a framework to leverage parallelism on exascale systems
- Comprises Aerospace, Chemistry, CS, ECE, Mechanical Eng
- Some of the tools being developed:
  - Moya: Just-in-time recompilations
  - Tangram: Compiler programming system for performance portability
  - AMPI: Model for coarse-grained overdecomposition for load balancing
  - PickPocket: Data relocation for efficient computation



Performance Optimization

- Applications are often targeting multiple complex systems → Large optimization space
- Compilers deliver unsatisfactory performance (-O3 is not enough)
  No Optimization → No Performance
- Hard to maintain and manage optimizations as the code evolves and new features are added
- And, as optimizations are added it becomes hard to maintain the code



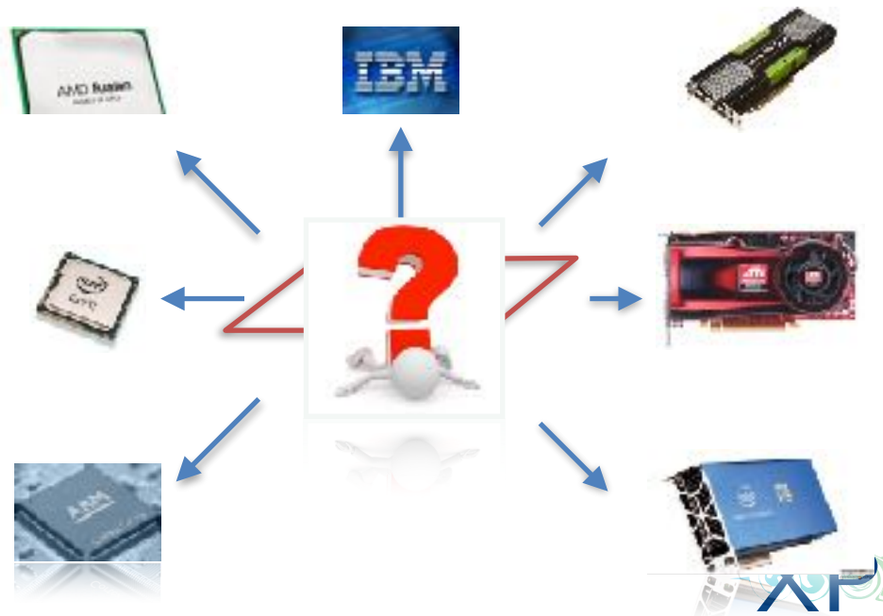
Performance Optimization on HPC

- Scientists make decisions based on maximizing scientific output, not application's performance
- They also want to control the performance to the level needed, even sacrificing abstraction and ease of programming
- A new technology that can coexist with older ones has a greater chance of success (e.g. MPI)
- No complete buy-in at the beginning
  - A barrier for new frameworks is that you can't integrate them incrementally
- Risk-mitigation strategy is to let competing technologies coexist, but not always possible

Source: *Understanding the High-Performance Computing Community: A Software Engineer's Perspective*. Victor Basili et al

Performance Optimization

- How handle all the required optimizations together for many different scenarios?
- How to keep the code maintainable?
- How to find the best sequence of optimizations?

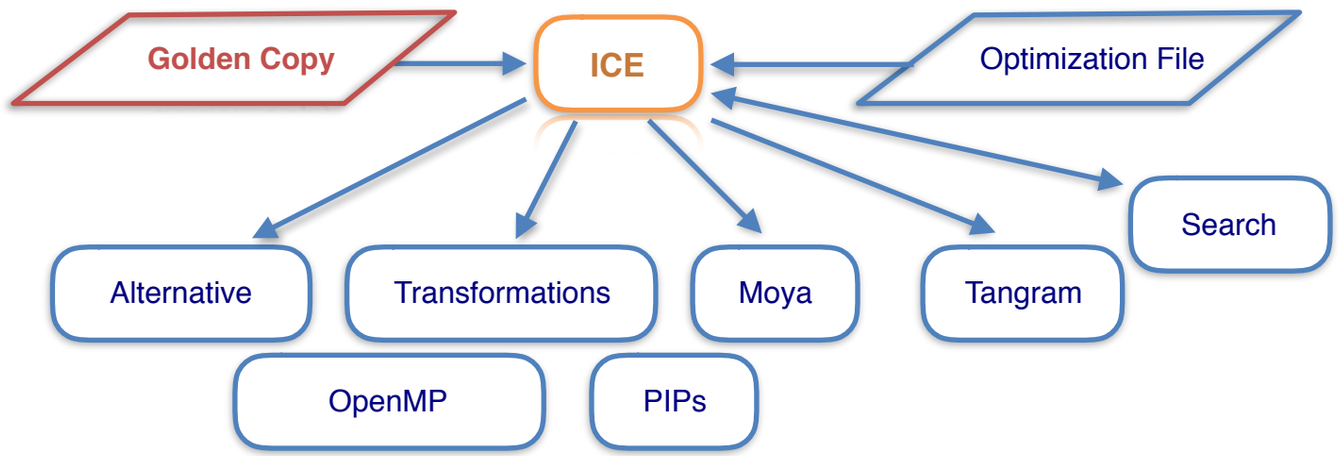


Goal

- No complete buy-in
- Incremental adoption
- Coexistence with other tools
- Automatically finding the best sequence of optimizations and applying them without disrupting the original code is important to improve performance and keep the code maintainable

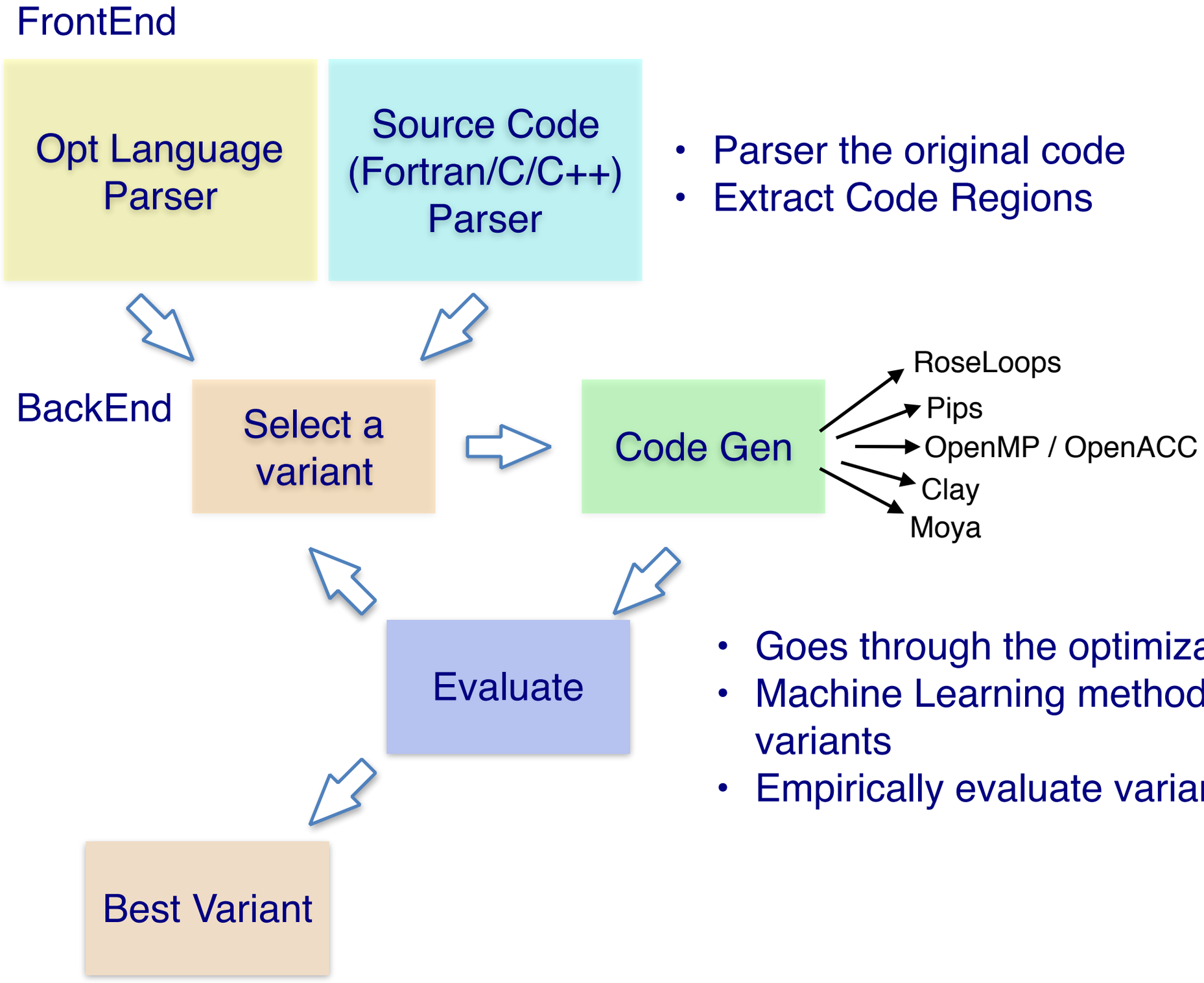
ICE

- Illinois Coding Environment
- Golden copy approach: baseline version without architecture- or compiler-specific optimizations (not buy-in)
- Search combined with application's developer expertise
- Build-time, Compile-time and Runtime optimizations
- Non-prescriptive, Gradual adoption, Separation of Concerns
- Reuse of other optimizations tools already implemented
  - Interfaces to simplify plug-in search and optimization tools



ICE

- Source code is annotated to define code regions
- Optimization file notation orchestrates the use of the optimization tools on the code regions defined
- Interface provides operations on the Source code to invoke optimizations through:
  - Adding pragmas
  - Adding labels
  - Replacing code regions
- These operations are used by the interface to plug-in optimization tools
- Most tools are source-to-source
  - tools must understand output of previous tools



- Parser the original code
- Extract Code Regions

- Goes through the optimization space
- Machine Learning methods to select variants
- Empirically evaluate variants

Optimization Tools

- Pips (MINES ParisTech)
  - Code optimization tool based on polyhedral framework
- Moya (Tarun Prabhu/UIUC)
  - Rutime optimizations
- Clay (Joël Poudroux, Oleksandr Zinenko)
  - Loop transformations using the polyhedral framework
- OpenMP
  - Parallelization of code regions using pragmas
- RoseLoop
  - Loop transformations based Rose compiler infrastructure
- Altdesc
  - Replace of code regions (e.g. hand optimized ones)

Search Methods

- Optimizations Space: optimizations, parameters and software stack (compiler version, flags, libraries)
- It cannot be exhaustively traversed (gcc parameters has 10^{806} configurations)
- Complex space that requires different search techniques
- OpenTuner (Jason Ansel et al)
  - Meta technique is used to control the use of the other techniques (e.g round robin, random, auc bandit)
  - Multi-armed bandit problem: deciding which, on which order, and how many times to pull levers on a slot machine with many arms with an unknown payout probability
- Spearmint (Jasper Snoek et al)
  - Bayesian Optimization of Machine Learning Algorithms (NIPS'12)

Optimization Language

- Domain specific (easier and straightforward)
- Expose the optimization space
  - What sequences of optimizations to evaluate?
  - What are the best parameters?
- Control the use of the optimization tools
- Record the steps to efficient code
- It can be shared with others or go along with the deployment and installation

ICE

Annotations in Fortran

- Block

```
! @ICE block=b1
  …
! @ICE endblock
```

- Loop

```
! @ICE loop=l1
  DO i = 1, n
   …
  END DO
```

Annotations in C/C++

- Block

```
#pragma @ICE block=<id>
     …
#pragma @ICE endblock
```

- Loop

```
#pragma @ICE loop=<id>
  for(…) {
   …
   }
```

ICE

- Optimization file (extended .yaml)
  - Direct
  - Search

```
---
<preamble commands>

<block/loop id>: <ref>
  <commands>[*+?]
...
```

- <commands>+ : 1 or more in the combinations
- <commands>* : 0 or more in the combinations
- <commands>? : 0 or 1

```
---
compilers: [gcc, icc]
# Built command before compilation
prebuildcmd:

# Compilation command before tests
buildcmd: make clean all

#Command call for each test
runcmd: time ./run.sh

search: on # or off
```

```
memoryBound: &id01
  - unroll:
      loop: 3
      factor: 4
  - tile:
      loop: 8
      factor: 1

example1: *id01
  - runtime:

example2:
  - altdesc: ./opt2/*.opt

sc2:
  - altdesc: ./opt2/*.opt

...
```

ICE

- Loop Interchange

```
---
L1:
  - interchange:
    order: 2,1,0
...
```

```
! @ICE loop=L1
do i = 1,n
   do j = 1,m
      a(i) = a(i) + b(i,j) * c(j)
   end do
end do
```

```
! @ICE loop=L1
do j = 1,m
   do i = 1,n
      a(i) = a(i) + b(i,j) * c(j)
   end do
end do
```

- Loop indices in the nest start from 0
- Order accepts * to generate all combinations
- Order accepts | to select specific combinations
  - Example: 2,1,0|1,0,2

ICE

- Loop Unroll

```
---
L2:
  - unroll:
    loop: 2
    factor: 3
...
```

- Loop indices in the nest start from 1
- Factor accepts .. to generate a range
  - Example: 2..10

```
! @ICE loop=L2
do i = 1,n
   do j = 1,m
      a(i) = a(i) + b(i,j) * c(j)
   end do
end do
```

```
! @ICE loop=L2
do i = 1,n
   do j = 1,m,3
      a(i) = a(i) + b(i,j)   * c(j)
      a(i) = a(i) + b(i,j+1) * c(j+1)
      a(i) = a(i) + b(i,j+2) * c(j+2)
   end do
end do
```

Cache Option

- The best variant found for each code region is saved in a cache
- Consistency: a hash of the golden copy's code region that it was based on is also saved along with the best variant
- Able to take advantage of the efficient generated code avoiding the installation of all the tool chain
- Cache can be shipped with the application and invoked according to the machine/system used

Evaluation

- Matrix Multiplication

```
#pragma @ICE loop=matmul
        for (i=0; i<matSize; i++)
            for (j=0; j<matSize; j++) {
                for (k=0; k<matSize; k++) {
                    matC[i][j] += matA[i][k] * matB[k][j];
                }
            }
        }
```

+

```
—
# Built command before compilation
prebuildcmd:

# Compilation command before tests
buildcmd: make realclean;  make

#Command call for each test
runcmd: ./mmc

matmul:
    - Pips.tiling+:
        loop: 1
        factor: [2..512, 2..512, 2..512]
    - Pips.tiling*:
        loop: 4
        factor: [8, 16, 8]
     - OpenMP.OMPFor*:
        loop: 1

…
```

Evaluation

- Matrix Multiplication

```
#pragma @ICE loop=matmul
      for (i=0; i<matSize; i++)
         for (j=0; j<matSize; j++) {
            for (k=0; k<matSize; k++) {
               matC[i][j] += matA[i][k] * matB[k][j];
            }
         }
      }
```

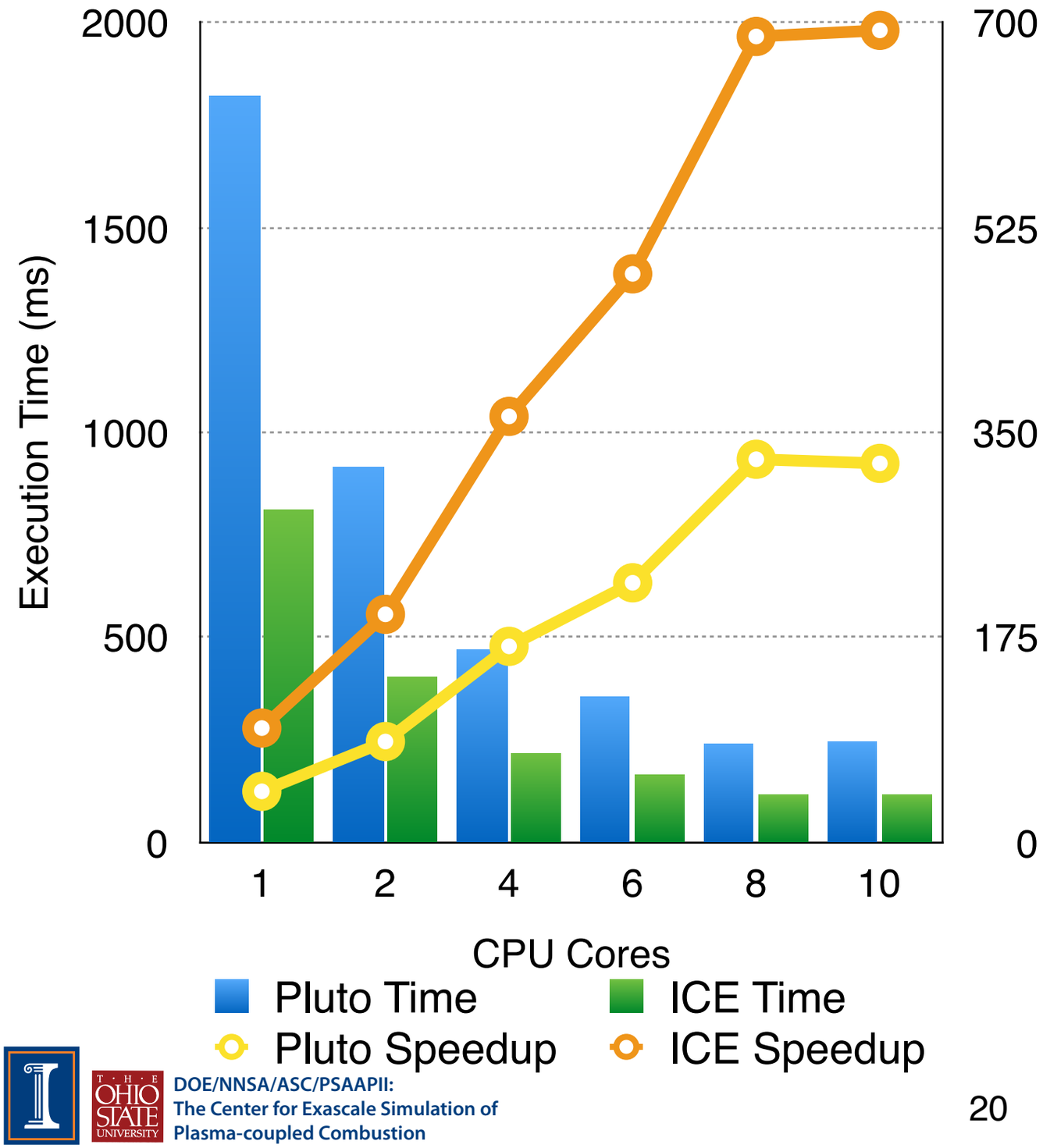
=

```
#pragma omp parallel for schedule(static,1) private(i_t, k_t, j_t,i_t_t, k_t_t ,j_t_t, i, k,j)
for (i_t = 0; i_t <= 127; i_t += 1)
  for (k_t = 0; k_t <= 127; k_t += 1)
  for (j_t = 0; j_t <= 3; j_t += 1)
  for (i_t_t = 4 * i_t; i_t_t <= ((4 * i_t) + 3); i_t_t += 1)
  for (k_t_t = 2 * k_t; k_t_t <= ((2 * k_t) + 1); k_t_t += 1)
  for (j_t_t = 32 * j_t; j_t_t <= ((32 * j_t) + 31); j_t_t += 1)
  for (i = 4 * i_t_t; i <= ((4 * i_t_t) + 3); i += 1)
  for (k = 8 * k_t_t; k <= ((8 * k_t_t) + 7); k += 1)
  for (j = 16 * j_t_t; j <= ((16 * j_t_t) + 15); j += 1)
  matC[i][j] += matA[i][k] * matB[k][j];
```

+

```
—
# Built command before compilation
prebuildcmd:

# Compilation command before tests
buildcmd: make realclean;  make

#Command call for each test
runcmd: ./mmc

matmul:
   - Pips.tiling+:
       loop: 1
      factor: [2..512, 2..512, 2..512]
   - Pips.tiling+:
      loop: 4
      factor: [8, 16, 8]
   - OpenMP.OMPFor+:
      loop: 1

…
```

Matrix Multiplication



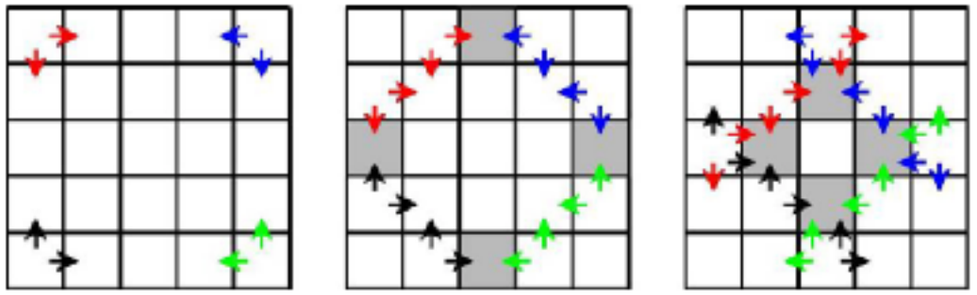
- Two levels of tiling + OpenMP
- Original version: 78,825 ms
- 98x speedup (1 core)
- 694x speedup (10 cores)
- Avg 2.2x speedup over Pluto

2048^2 elements
icc 17.0.1
Intel E5-2660 v3
Pluto Pet branch

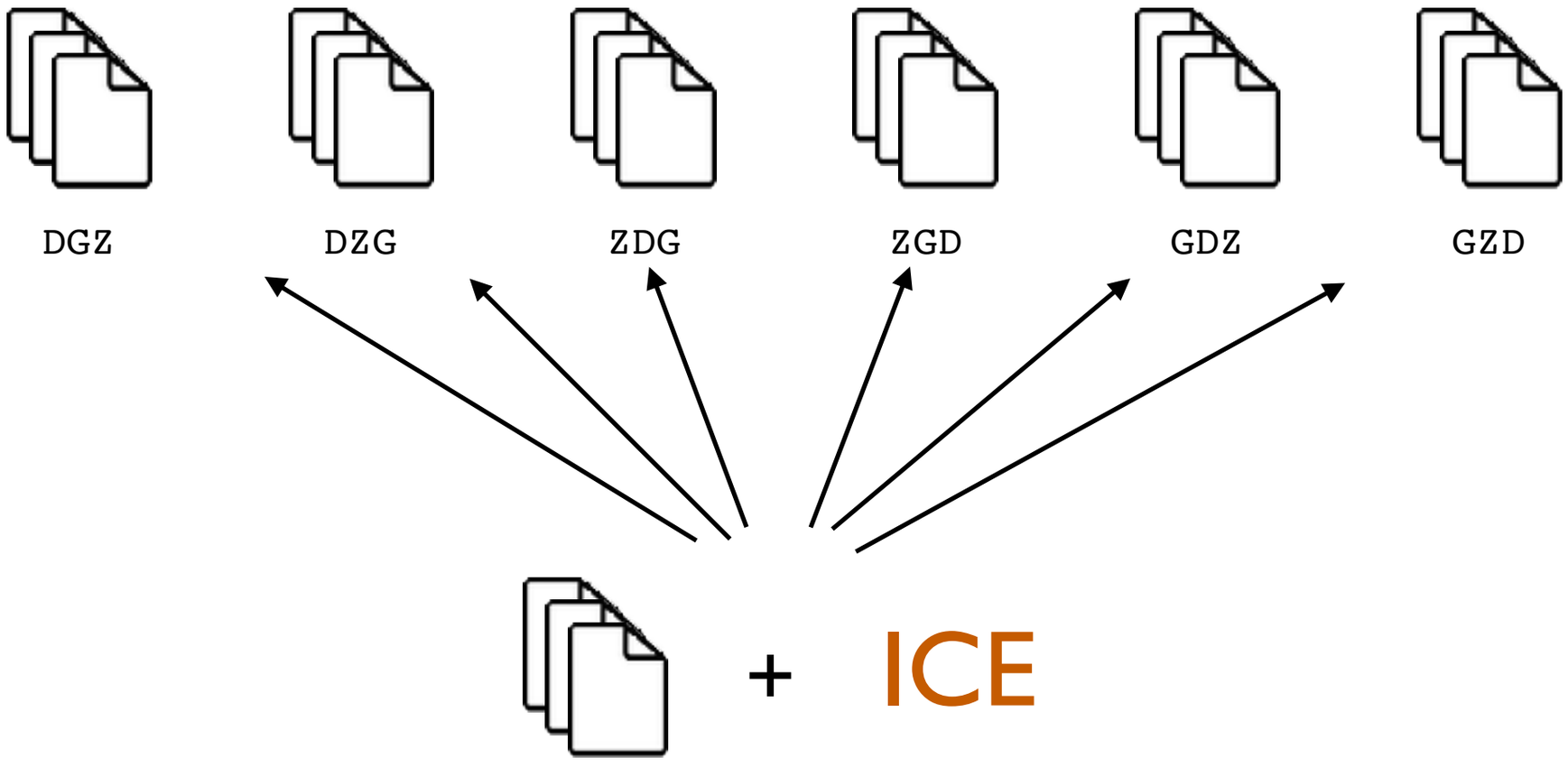
DOE/NNSA/ASC/PSAAPII:
The Center for Exascale Simulation of
Plasma-coupled Combustion

XPACC

Kripke



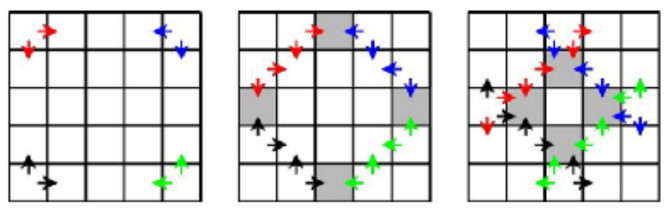
- 3D Sn deterministic particle transport code.
- Proxy-App for the LLNL transport code ARDRA
- 6 data layouts of angular fluxes (Psi) and 6 hand-tuned versions



*Developed by Adam J. Kunen, Peter N. Brown, Teresa S. Bailey, Peter G. Maginot.
Source: https://codesign.llnl.gov/kripke.php

Experiments

- LLNL Kripke + ICE
  - Optimizations according to the data layout selected
  - Loop transformations + inline data layout access
  - 10-30% slower than hand-optimized, but 6-8x speedup over baseline
  - Other optimizations under test to close the hand-optimized performance gap



Conclusions

- In order to harness all computing power available in current and future architectures, it is necessary to apply specific optimizations to the source code
- ICE:
  - Separation of Concerns (opt file) +
  - Coexistence with other tools +
  - Gradual adoption +
  - Empirical search + Developer Knowledge
- Golden copy: the developer can focus on the problem
- Simple and easy to be used by the programmers
- Hard to get the tools to work though!

This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number DE-NA0002374.