Effective Tuning of Automatically Parallelized OpenMP Applications Using The State of the art Parallel Optimizing Compiler.

Ph.D Candidate Miguel Romero Rosas and
Dr. Rudolf Eigenmann

CGO 2025

University of Delaware

Introduction

- The demand for efficient parallel programming persists.
- Automatic Parallelization VS Hand Parallelization.
- Inability to make optimize choices at compile time.
- Insufficient knowledge of target applications.

Can effective tuning techniques in Automatic Parallelizers enable applications to consistently match or exceed the performance of manually parallelized implementations?

Important Contributions

Present a Tuning study of the state of the art Parallelizer Compiler called Cetus, utilizing study cases from real-world applications, such as the NAS Parallel Benchmarks Suite (NPB) v3.3 , the POLYBENCHMARK PB) Suite v4.2.

Important Contributions

Present a Tuning study of the state of the art Parallelizer Compiler called Cetus, utilizing study cases from real-world applications, such as the NAS Parallel Benchmarks Suite (NPB) v3.3 , the POLYBENCHMARK PB) Suite v4.2.

Introduced a novel Portable Tuning Framework (PTF) v1.0 that optimizes different program sections at once.

Important Contributions

Present a Tuning study of the state of the art Parallelizer Compiler called Cetus, utilizing study cases from real-world applications, such as the NAS Parallel Benchmarks Suite (NPB) v3.3 , the POLYBENCHMARK PB) Suite v4.2.

Introduced a novel Portable Tuning Framework (PTF) v1.0 that optimizes different program sections at once.

Present a evaluation performance among two different compilers , GCC and Clang.

Motivation Problem

- Optimizations depends on the program and the target platform.
- The different sets of optimizations will create a vast optimization space.

```
#pragma experimental start
for (k=0; k<=(grid_points[2]-1); k ++ )
 {
      ...computation
 }
#pragma experimental end
```

Motivation Problem

- Optimizations depends on the program and the target platform.
- The different sets of optimizations will create a vast optimization space.

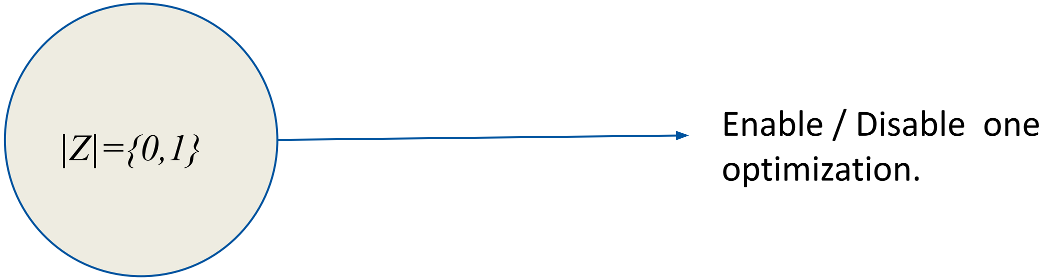
```
#pragma experimental start
for (k=0; k<=(grid_points[2]-1); k ++ )
 {
      ...computation
 }
#pragma experimental end
```

Parallelization
Loop Interchange
Array Reduction
..
.
.

Motivation Problem

- Optimizations depends on the program and the target platform.
- The different sets of optimizations will create a vast optimization space.

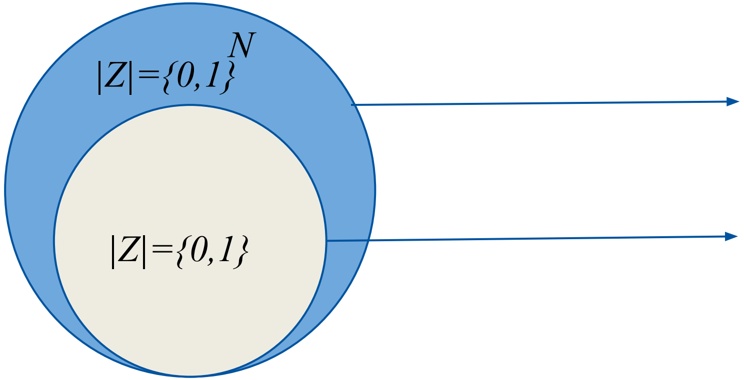
```
#pragma experimental start
for (k=0; k<=(grid_points[2]-1); k ++ )
 {
     ...computation
 }
#pragma experimental end
```



Motivation Problem

- Optimizations depends on the program and the target platform.
- The different sets of optimizations will create a vast optimization space.

```
#pragma experimental start
for (k=0; k<=(grid_points[2]-1); k ++ )
 {
      ...computation
 }
#pragma experimental end
```



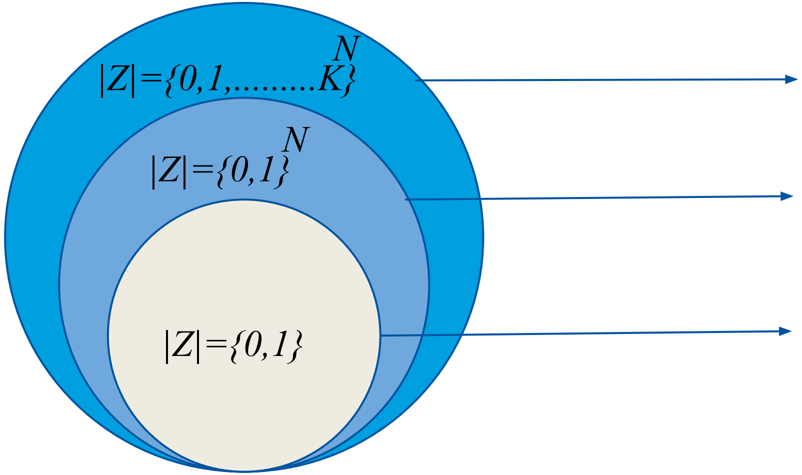
Enable / Disable N type of optimations.

Enable / Disable one optimization.

Motivation Problem

- Optimizations depends on the program and the target platform.
- The different sets of optimizations will create a vast optimization space.

```
#pragma experimental start
for (k=0; k<=(grid_points[2]-1); k ++ )
 {
.      ...computation
 }
#pragma experimental end
```



Different SubLevels of the different optimizations.

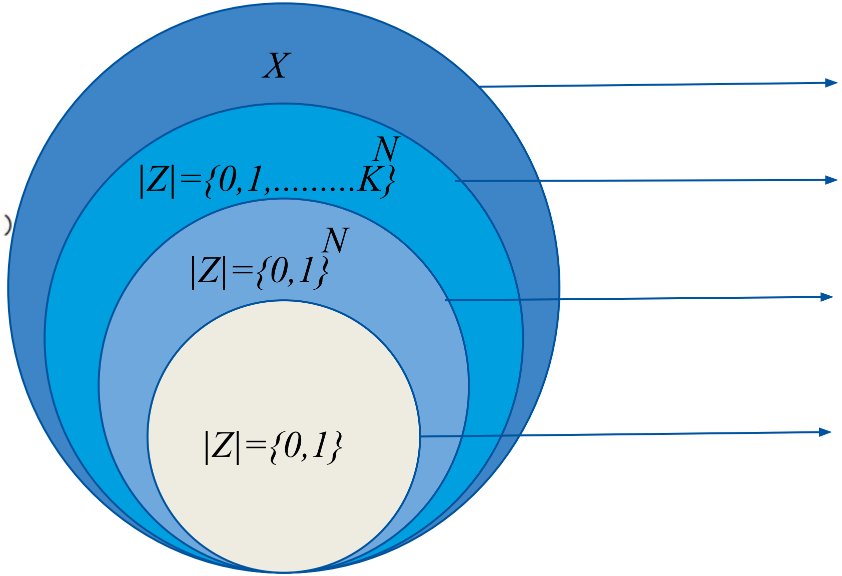
Enable / Disable N type of optimations.

Enable / Disable one optimization.

Motivation Problem

- Optimizations depends on the program and the target platform.
- The different sets of optimizations will create a vast optimization space.

```
#pragma experimental start
for (k=0; k<=(grid_points[2]-1); k ++ )
 {
      ...computation
 }
#pragma experimental end
```



The space of potential combinations of optimizations.

Different SubLevels of the different optimizations.

Enable / Disable N type of optimations.

Enable / Disable one optimization.

Challenge Search Space

- Evaluating many optimization variants and choosing the one that performs the best at runtime.
- The effect of an optimization may depend significantly on the presence of another.
- Interactions between the optimization variants. (Windows Size).

Challenge Search Space

- Evaluating many optimization variants and choosing the one that performs the best at runtime.
- The effect of an optimization may depend significantly on the presence of another.
- Interactions between the optimization variants. (Windows Size).

Challenge Search Space

- Evaluating many optimization variants and choosing the one that performs the best at runtime.
- The effect of an optimization may depend significantly on the presence of another.
- Interactions between the optimization variants. (Windows Size).

kernel #6

Challenge Search Space

- Evaluating many optimization variants and choosing the one that performs the best at runtime.
- The effect of an optimization may depend significantly on the presence of another.
- Interactions between the optimization variants. (Windows Size).

kernel #6

Loop #1

Loop #2

Loop #3

Loop #4

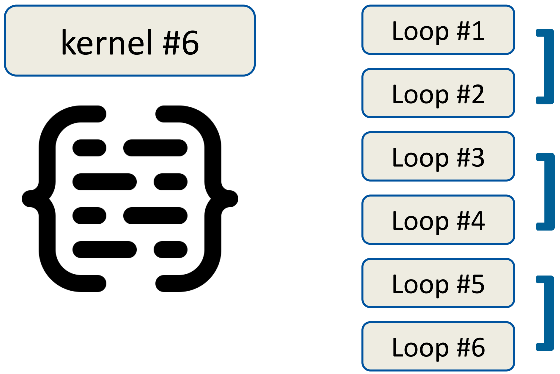
Loop #5

Loop #6

Challenge Search Space

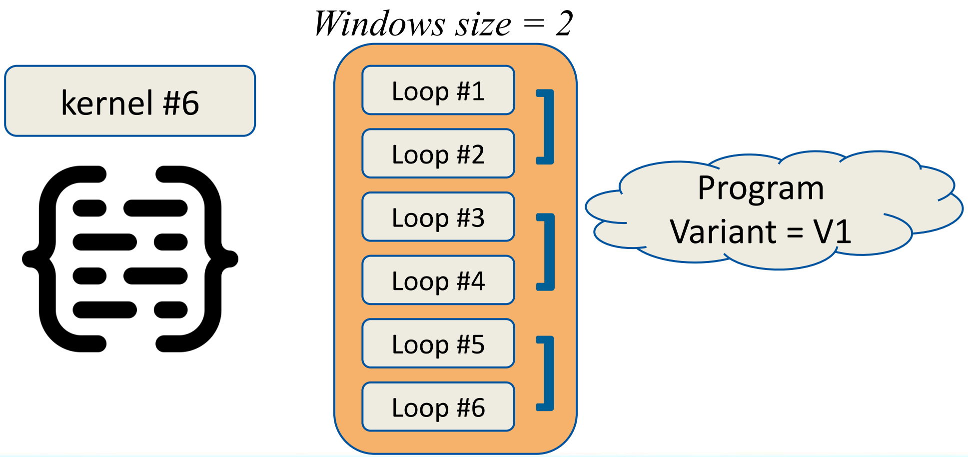
- Evaluating many optimization variants and choosing the one that performs the best at runtime.
- The effect of an optimization may depend significantly on the presence of another.
- Interactions between the optimization variants. (Windows Size).

Windows size = 2



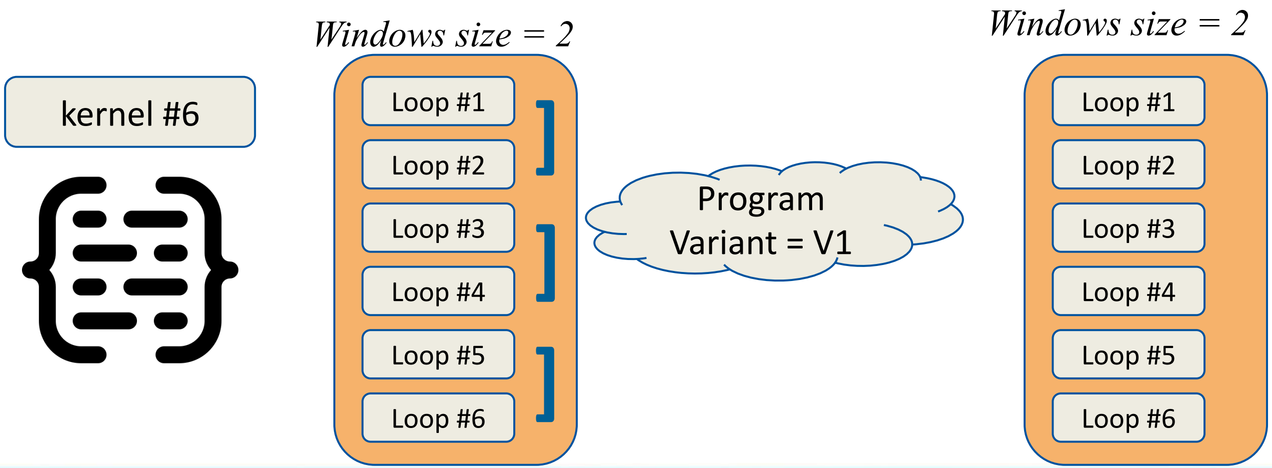
Challenge Search Space

- Evaluating many optimization variants and choosing the one that performs the best at runtime.
- The effect of an optimization may depend significantly on the presence of another.
- Interactions between the optimization variants. (Windows Size).



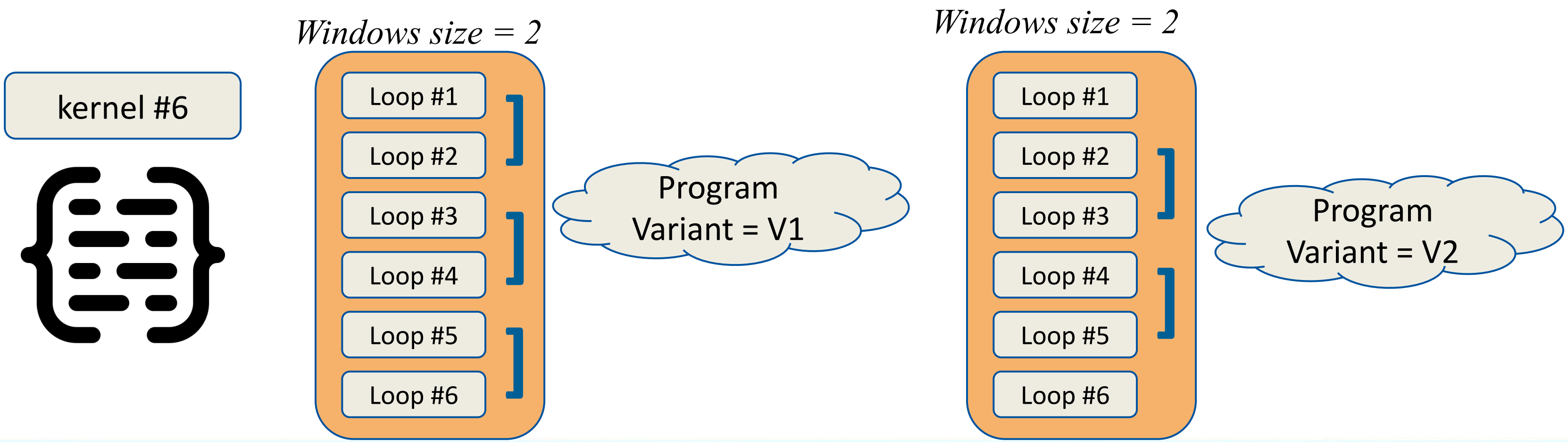
Challenge Search Space

- Evaluating many optimization variants and choosing the one that performs the best at runtime.
- The effect of an optimization may depend significantly on the presence of another.
- Interactions between the optimization variants. (Windows Size).



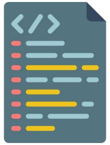
Challenge Search Space

- Evaluating many optimization variants and choosing the one that performs the best at runtime.
- The effect of an optimization may depend significantly on the presence of another.
- Interactions between the optimization variants. (Windows Size).

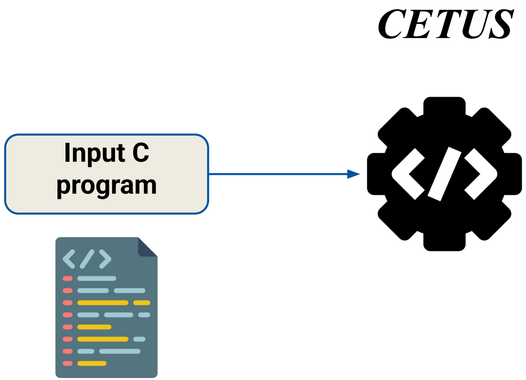


The Cetus Automatic Parallelizer

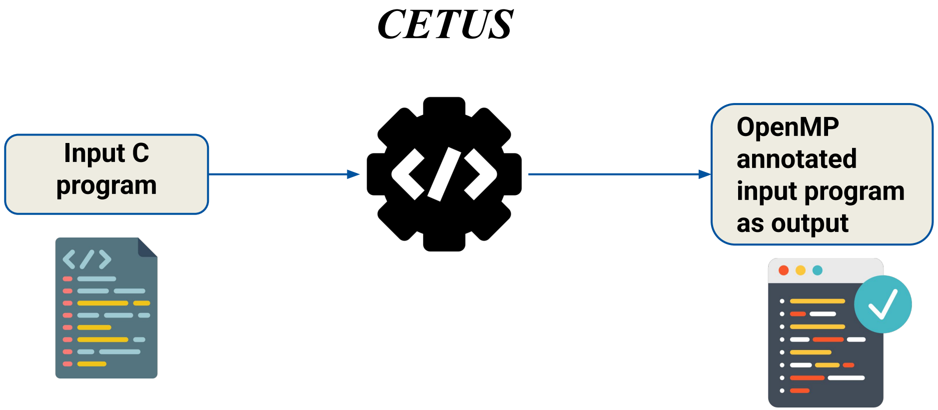
Input C program



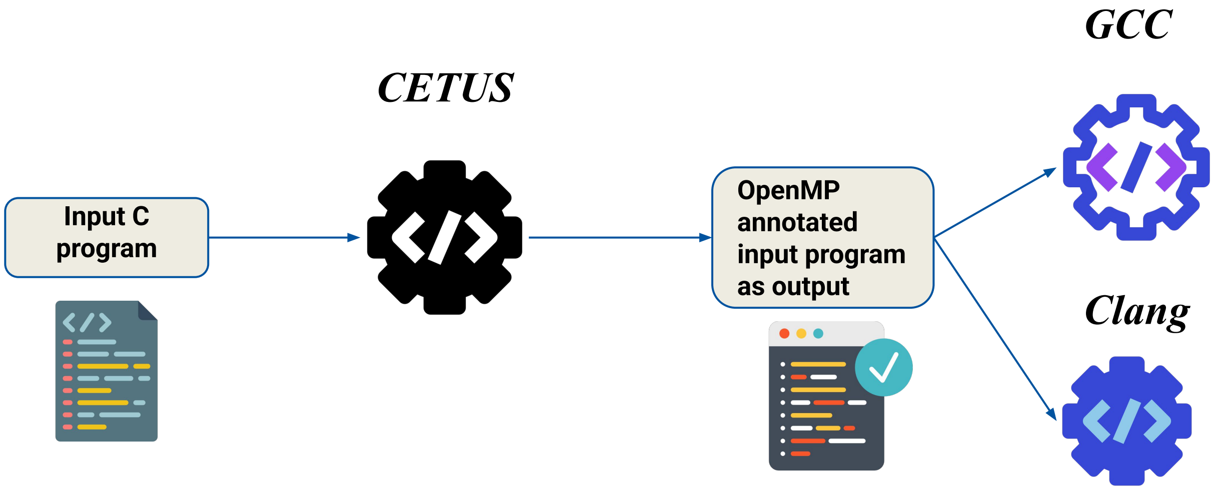
The Cetus Automatic Parallelizer



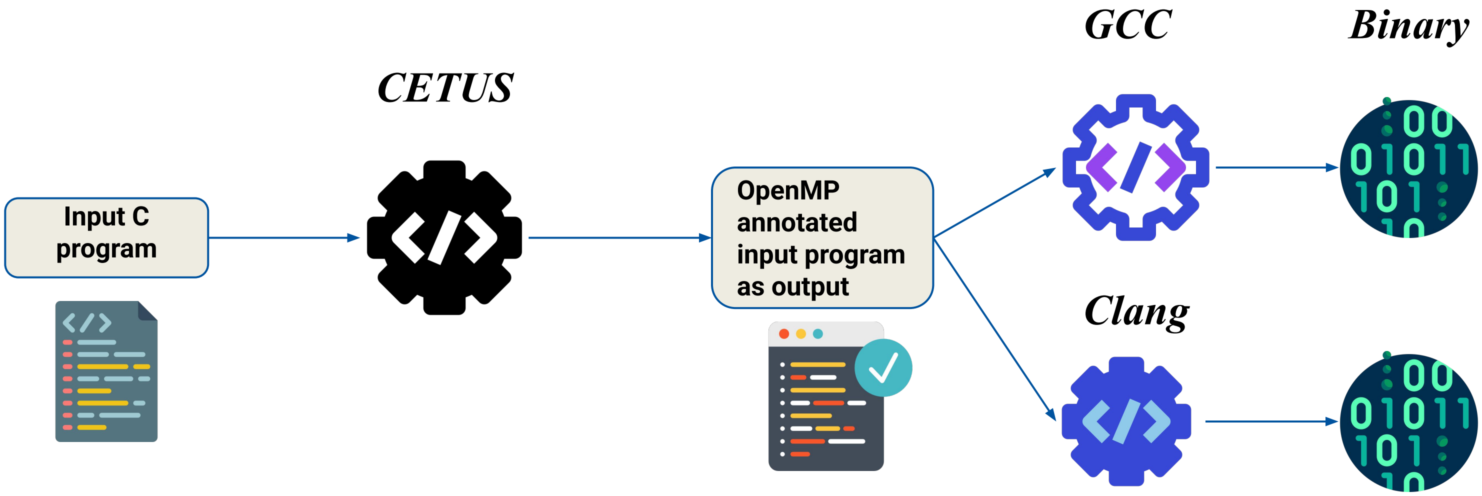
The Cetus Automatic Parallelizer



The Cetus Automatic Parallelizer



The Cetus Automatic Parallelizer



Portable Tuning Framework (PTF) V1.0

1. Search Space Navigation

2. Version creation

3. Evaluation

UNIVERSITY OF DELAWARE

Portable Tuning Framework (PTF) V1.0

Iteratively traverse the space

1. Search Space Navigation

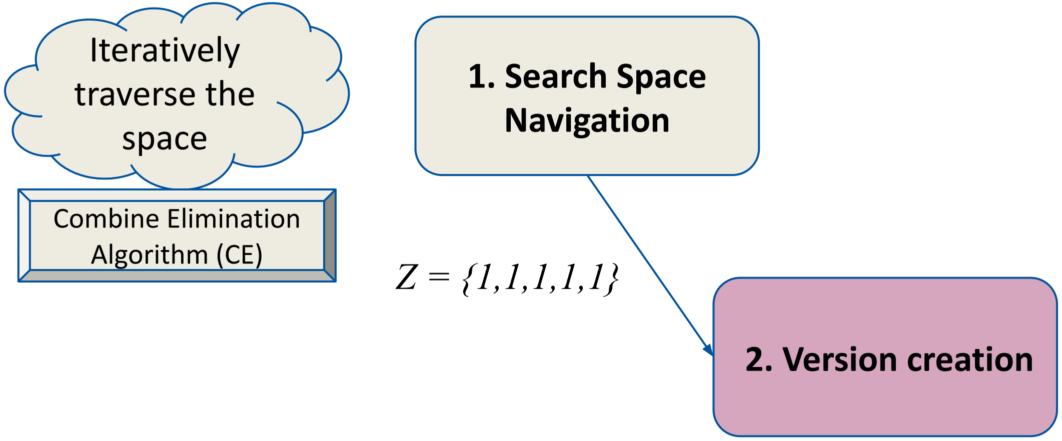
Portable Tuning Framework (PTF) V1.0

Iteratively traverse the space

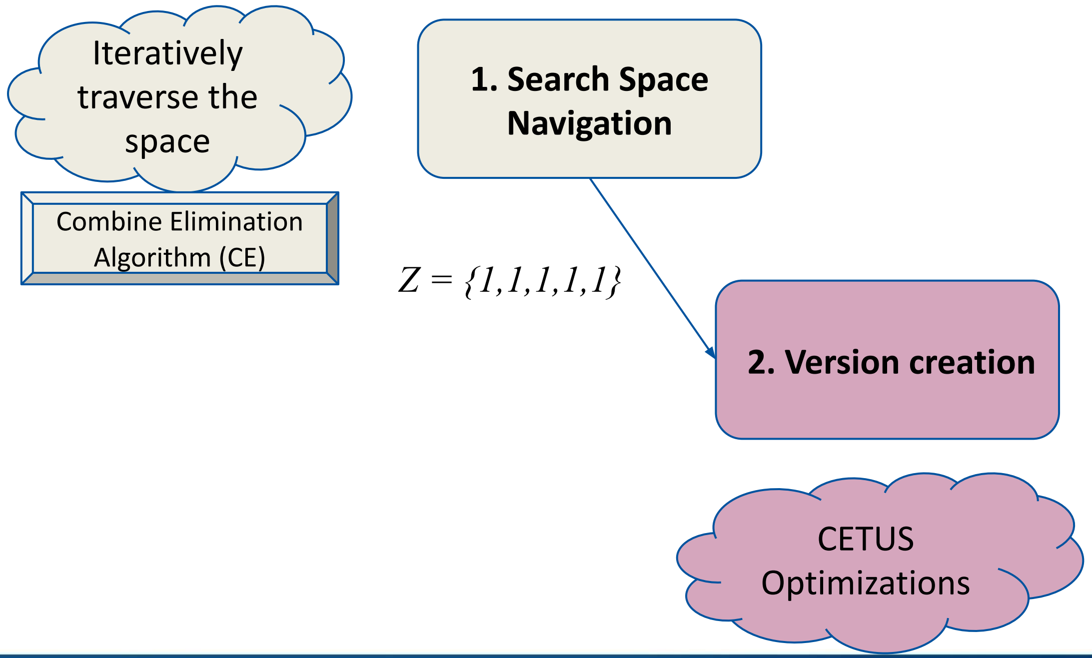
Combine Elimination Algorithm (CE)

1. Search Space Navigation

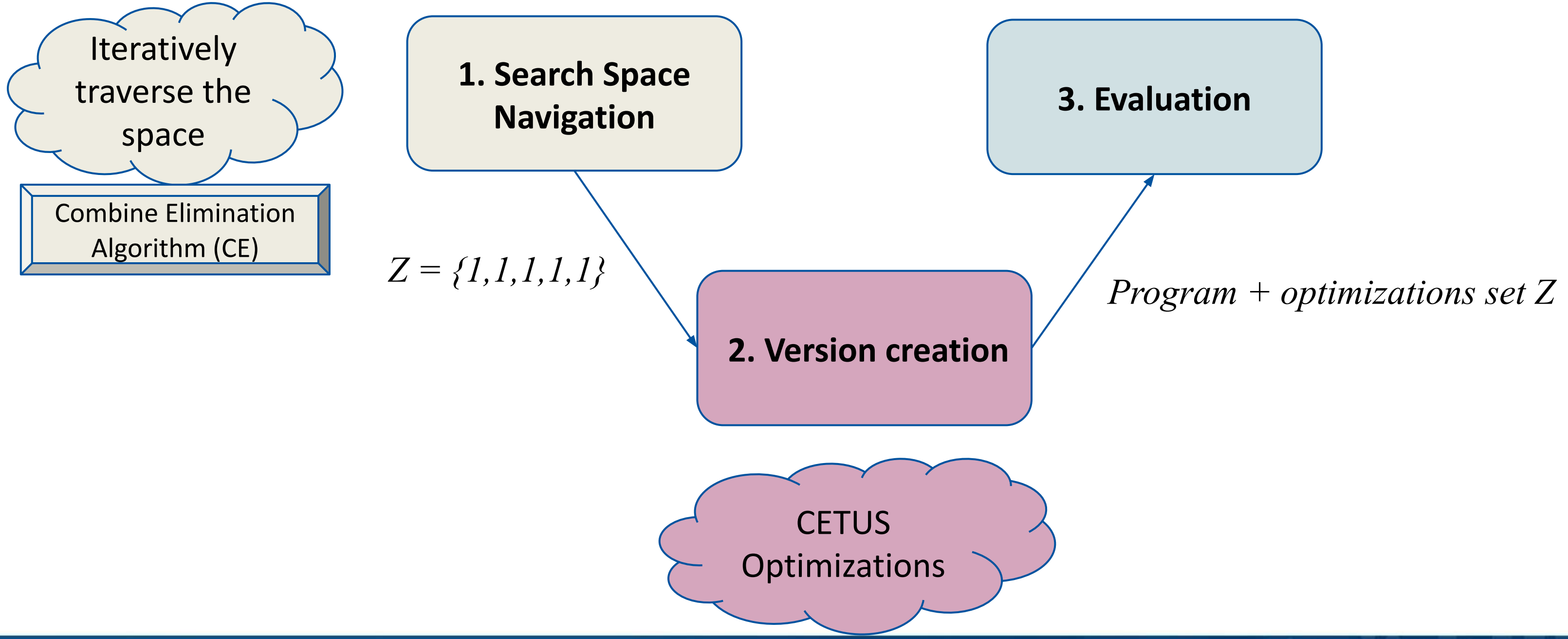
Portable Tuning Framework (PTF) V1.0



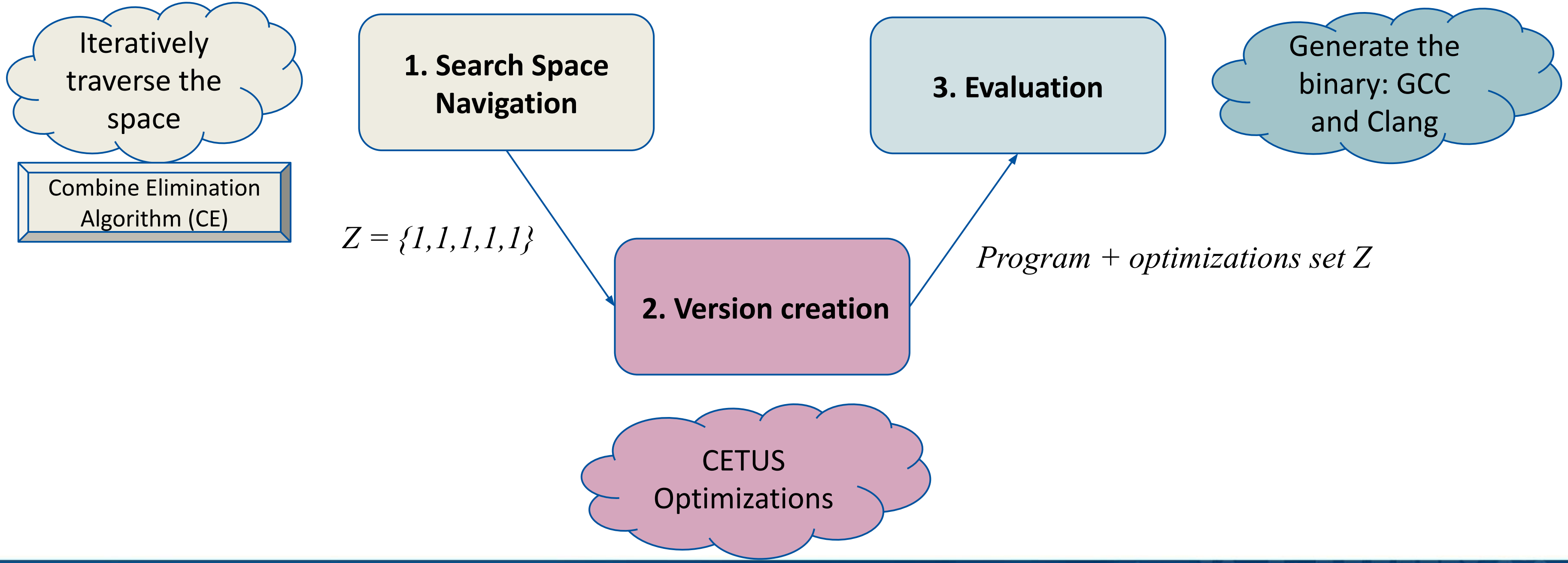
Portable Tuning Framework (PTF) V1.0



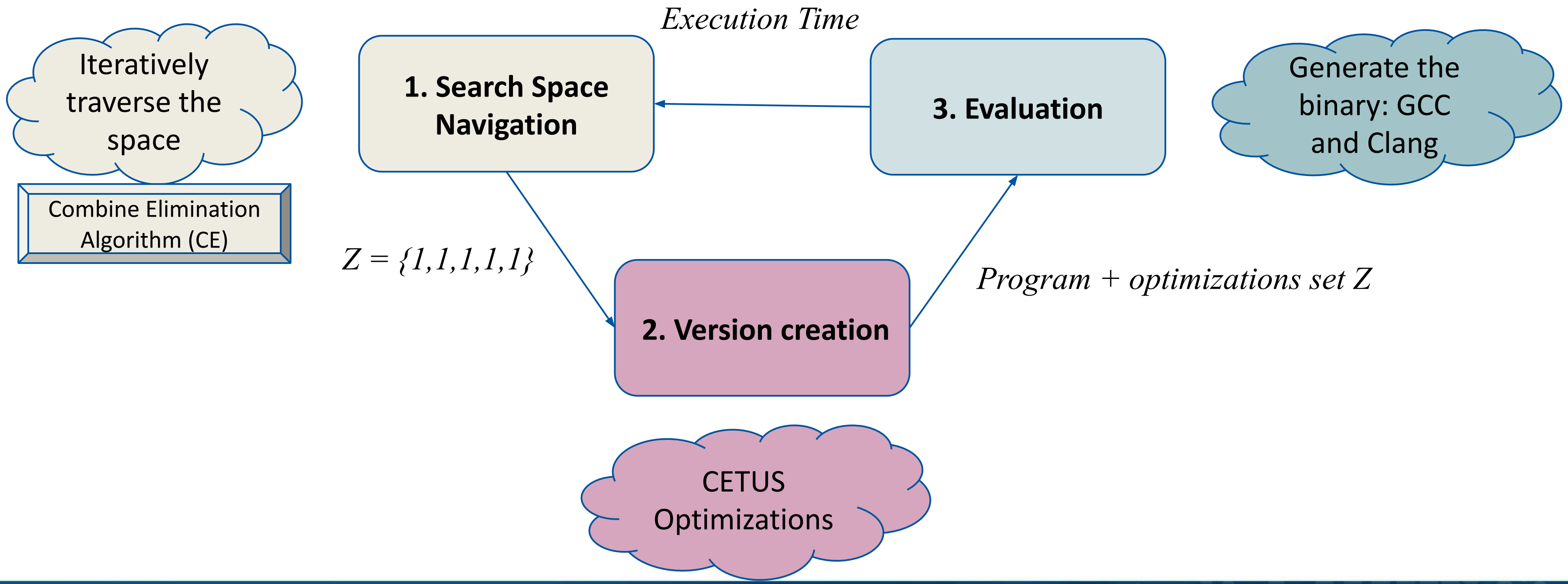
Portable Tuning Framework (PTF) V1.0



Portable Tuning Framework (PTF) V1.0



Portable Tuning Framework (PTF) V1.0



Combine Elimination Algorithm (CE)

- B = Baseline option combination

Combine Elimination Algorithm (CE)

- B = Baseline option combination
- S = Represent the optimization search space

Combine Elimination Algorithm (CE)

- B = Baseline option combination
- S = Represent the optimization search space
- S = {F1, F2, ..., Fn} and B = {F1 = 1, F2 = 1, ..., Fn = 1}.

Combine Elimination Algorithm (CE)

- B = Baseline option combination
- S = Represent the optimization search space
- S = {F1, F2, ..., Fn} and B = {F1 = 1, F2 = 1, ..., Fn = 1}.
- TB= Baseline execution time

Combine Elimination Algorithm (CE)

- B = Baseline option combination
- S = Represent the optimization search space
- S = {F1, F2, ..., Fn} and B = {F1 = 1, F2 = 1, ..., Fn = 1}.
- TB= Baseline execution time
- RIP = Relative Improvement Percentage

Combine Elimination Algorithm (CE)

- B = Baseline option combination
- S = Represent the optimization search space
- S = {F1, F2, ..., Fn} and B = {F1 = 1, F2 = 1, ..., Fn = 1}.
- TB= Baseline execution time
- RIP = Relative Improvement Percentage

$$RIP(F_i) = \frac{T(F_i = 0) - T(F_i = 1)}{T(F_i = 1)} \times 100\% \qquad (1)$$

UNIVERSITY OF DELAWARE

Combine Elimination Algorithm (CE)

- B = Baseline option combination
- S = Represent the optimization search space
- S = {F1, F2, ..., Fn} and B = {F1 = 1, F2 = 1, ..., Fn = 1}.
- TB= Baseline execution time
- RIP = Relative Improvement Percentage

$$RIP(F_i) = \frac{T(F_i = 0) - T(F_i = 1)}{T(F_i = 1)} \times 100\% \quad (1)$$

Execute the B ={All on}.

Combine Elimination Algorithm (CE)

- B = Baseline option combination
- S = Represent the optimization search space
- S = {F1, F2, ..., Fn} and B = {F1 = 1, F2 = 1, ..., Fn = 1}.
- TB= Baseline execution time
- RIP = Relative Improvement Percentage

$$RIP(F_i) = \frac{T(F_i = 0) - T(F_i = 1)}{T(F_i = 1)} \times 100\% \quad (1)$$

Execute the B ={All on}.

```
for each Fi in S:{
    RIP[Fi] = measureRIP(B, Fi);
}
```

Combine Elimination Algorithm (CE)

- B = Baseline option combination
- S = Represent the optimization search space
- S = {F1, F2, ..., Fn} and B = {F1 = 1, F2 = 1, ..., Fn = 1}.
- TB= Baseline execution time
- RIP = Relative Improvement Percentage

$$RIP(F_i) = \frac{T(F_i = 0) - T(F_i = 1)}{T(F_i = 1)} \times 100\% \quad (1)$$

Execute the B ={All on}.

```
for each Fi in S:{
    RIP[Fi] = measureRIP(B, Fi);
}
```

Identify the Most Negative RIP

Combine Elimination Algorithm (CE)

- B = Baseline option combination
- S = Represent the optimization search space
- S = {F1, F2, ..., Fn} and B = {F1 = 1, F2 = 1, ..., Fn = 1}.
- TB= Baseline execution time
- RIP = Relative Improvement Percentage

$$RIP(F_i) = \frac{T(F_i = 0) - T(F_i = 1)}{T(F_i = 1)} \times 100\% \quad (1)$$

Execute the B ={All on}.

```
1: for each Fi in S:{
2:     RIP[Fi] = measureRIP(B, Fi);
3: }
```

Identify the Most Negative RIP

B[MostRip] = 0

Combine Elimination Algorithm (CE)

- B = Baseline option combination
- S = Represent the optimization search space
- S = {F1, F2, ..., Fn} and B = {F1 = 1, F2 = 1, ..., Fn = 1}.
- TB= Baseline execution time
- RIP = Relative Improvement Percentage

$$RIP(F_i) = \frac{T(F_i = 0) - T(F_i = 1)}{T(F_i = 1)} \times 100\% \quad (1)$$

Execute the B ={All on}.

```
for each Fi in S:{
    RIP[Fi] = measureRIP(B, Fi);
}
```

Identify the Most Negative RIP

B[MostRip] = 0

S= S-B[MostRip]

Combine Elimination Algorithm (CE)

- B = Baseline option combination
- S = Represent the optimization search space
- S = {F1, F2, ..., Fn} and B = {F1 = 1, F2 = 1, ..., Fn = 1}.
- TB= Baseline execution time
- RIP = Relative Improvement Percentage

$$RIP(F_i) = \frac{T(F_i = 0) - T(F_i = 1)}{T(F_i = 1)} \times 100\% \qquad (1)$$

Execute the B ={All on}.

```
for each Fi in S:{
    RIP[Fi] = measureRIP(B, Fi);
}
```

Identify the Most Negative RIP

B[MostRip] = 0

S= S-B[MostRip]

Combine Elimination Algorithm (CE)

- B = Baseline option combination
- S = Represent the optimization search space
- S = {F1, F2, ..., Fn} and B = {F1 = 1, F2 = 1, ..., Fn = 1}.
- TB= Baseline execution time
- RIP = Relative Improvement Percentage

$$RIP(F_i) = \frac{T(F_i = 0) - T(F_i = 1)}{T(F_i = 1)} \times 100\% \qquad (1)$$

Execute the B ={All on}.

```
for each Fi in S:{
    RIP[Fi] = measureRIP(B, Fi);
}
```

Identify the Most Negative RIP

B[MostRip] = 0

S= S-B[MostRip]

Experimental Setup

- Our study compares two state-of-the-art optimizing compilers:

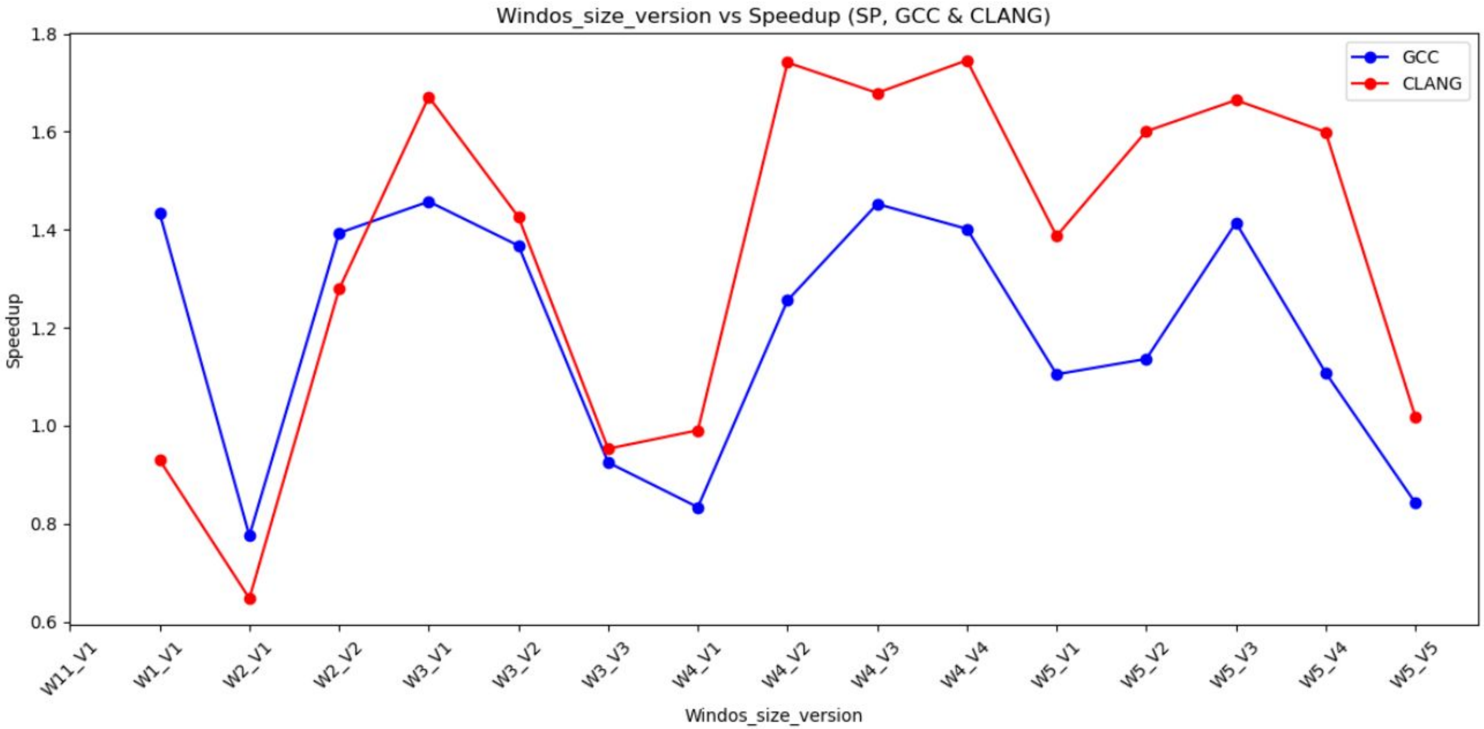
GCC	12.2.0
Clang	17.0.6

- Performance of the applications were measured:
  - ❑ CLASS B for NAS and LARGE_DATASET for the PB.
  - ❑ 16 Cores on a compute node featuring an INtel Xeon Gold 6230 processor
  - ❑ -O3 in each compiler
  - ❑ 8 different optimizations within Cetus

Preliminary Results

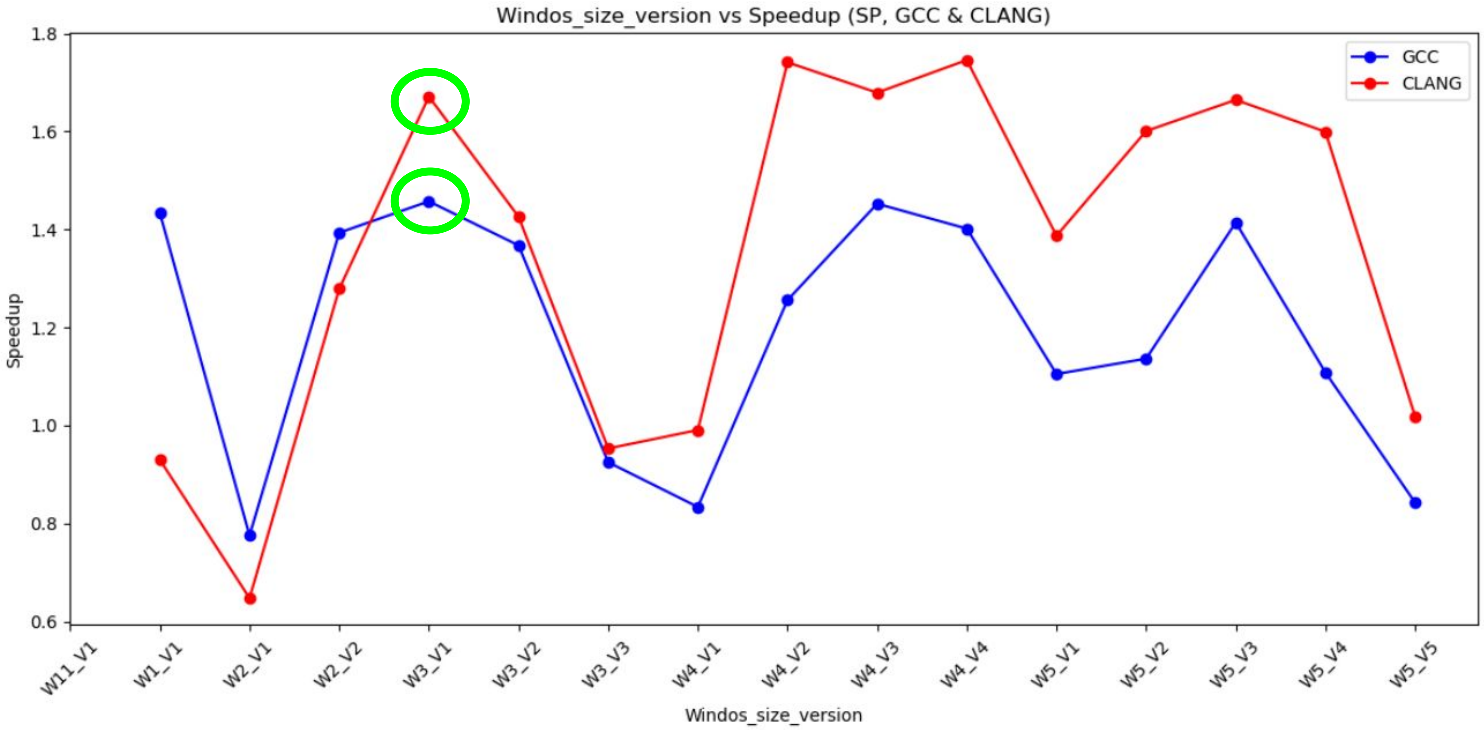
Windows Size Performance

Application SP from the Nas Parallel Benchmark suite. Subroutine Compute_rhs



Windows Size Performance

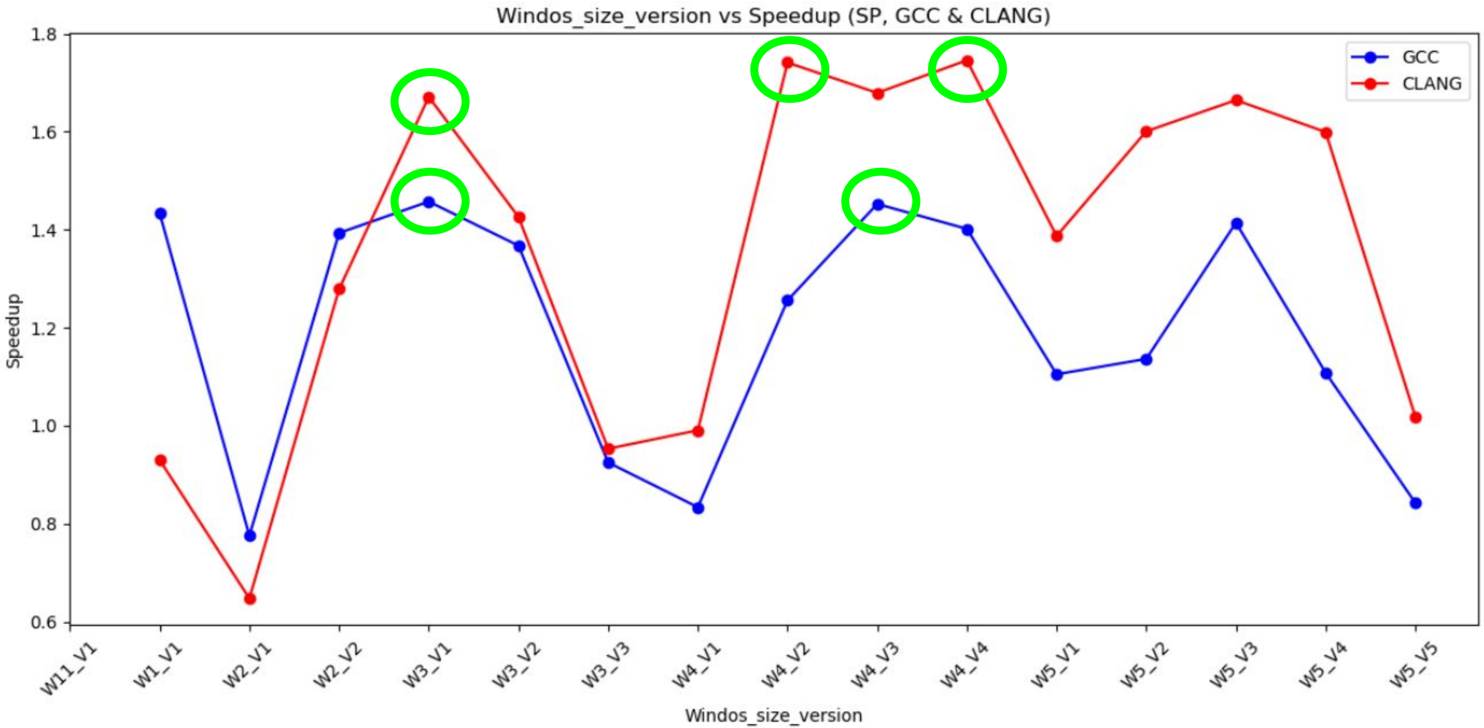
Application SP from the Nas Parallel Benchmark suite. Subroutine Compute_rhs



UNIVERSITY OF DELAWARE

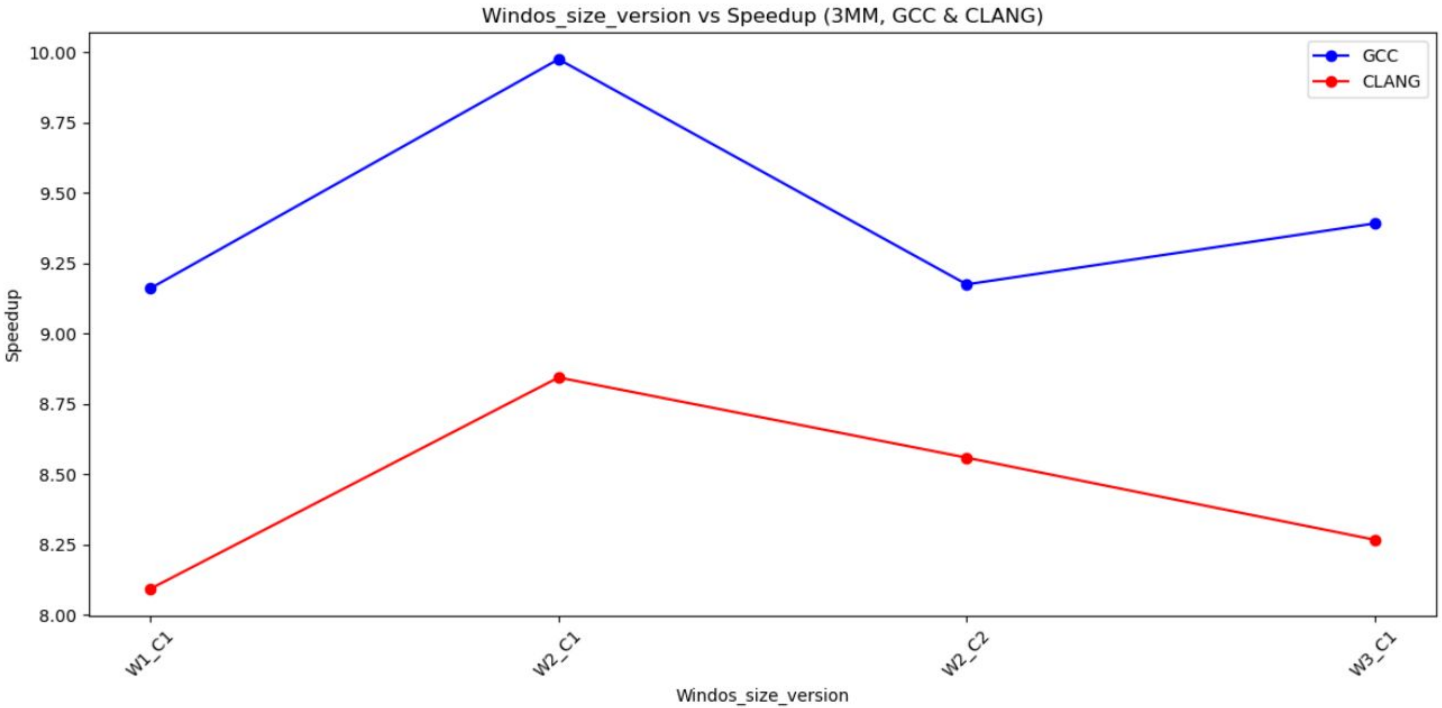
Windows Size Performance

Application SP from the Nas Parallel Benchmark suite. Subroutine Compute_rhs



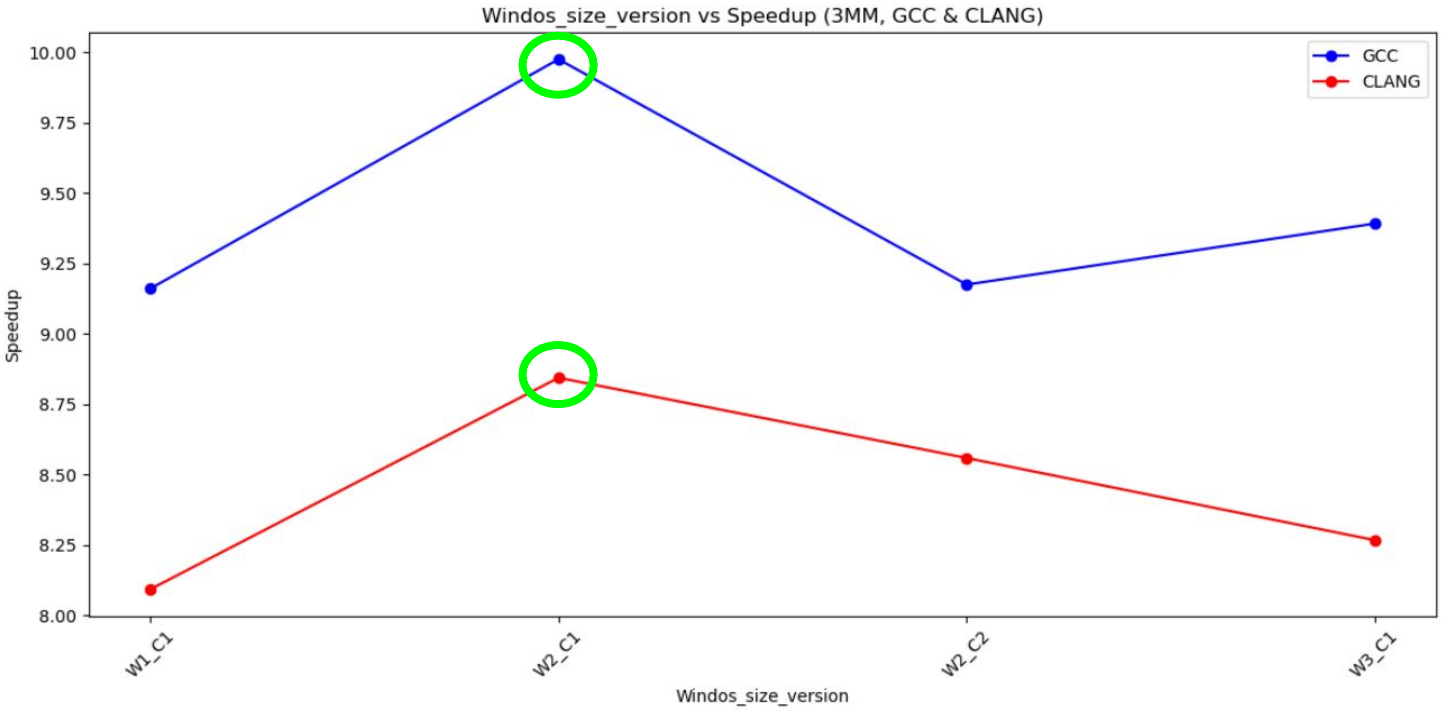
Windows Size Performance

Application 3MM from the Poly Benchmark suite. Subroutine Kernel_3MM



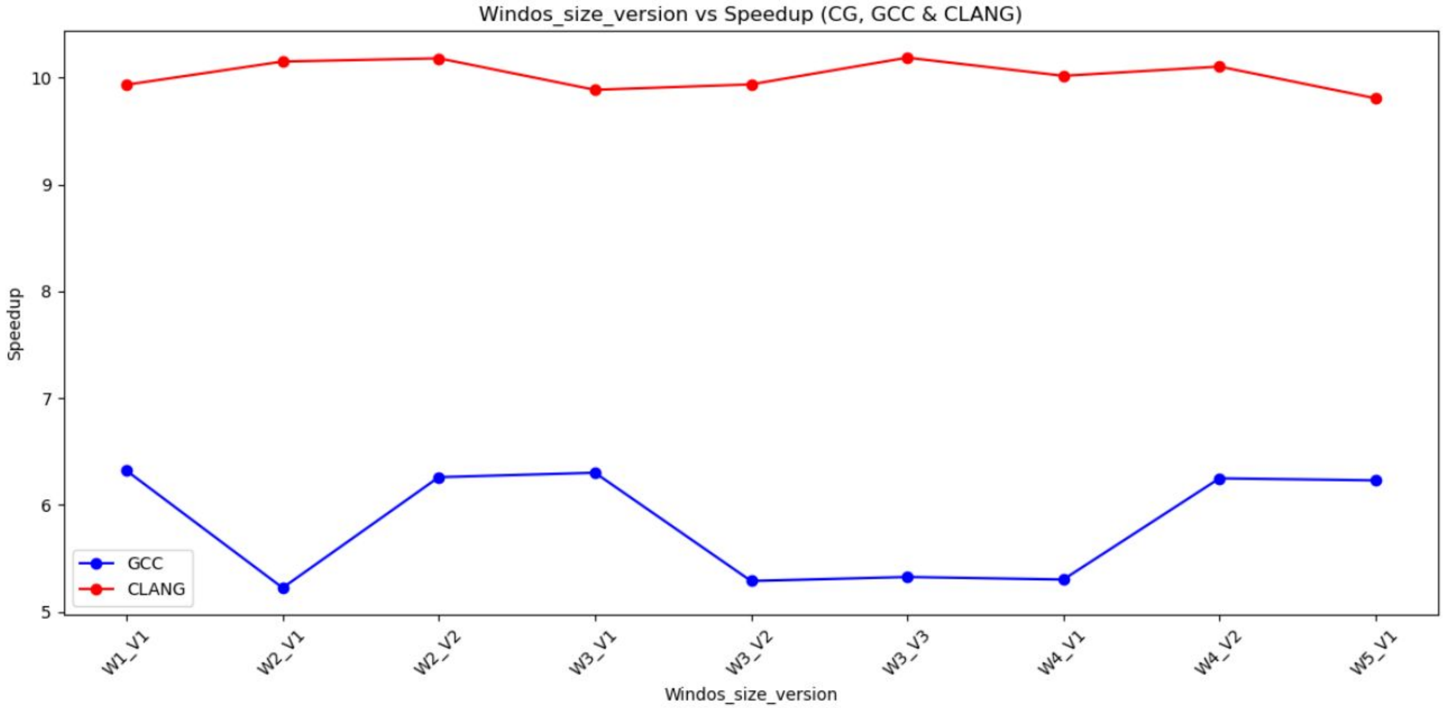
Windows Size Performance

Application 3MM from the Poly Benchmark suite. Subroutine Kernel_3MM



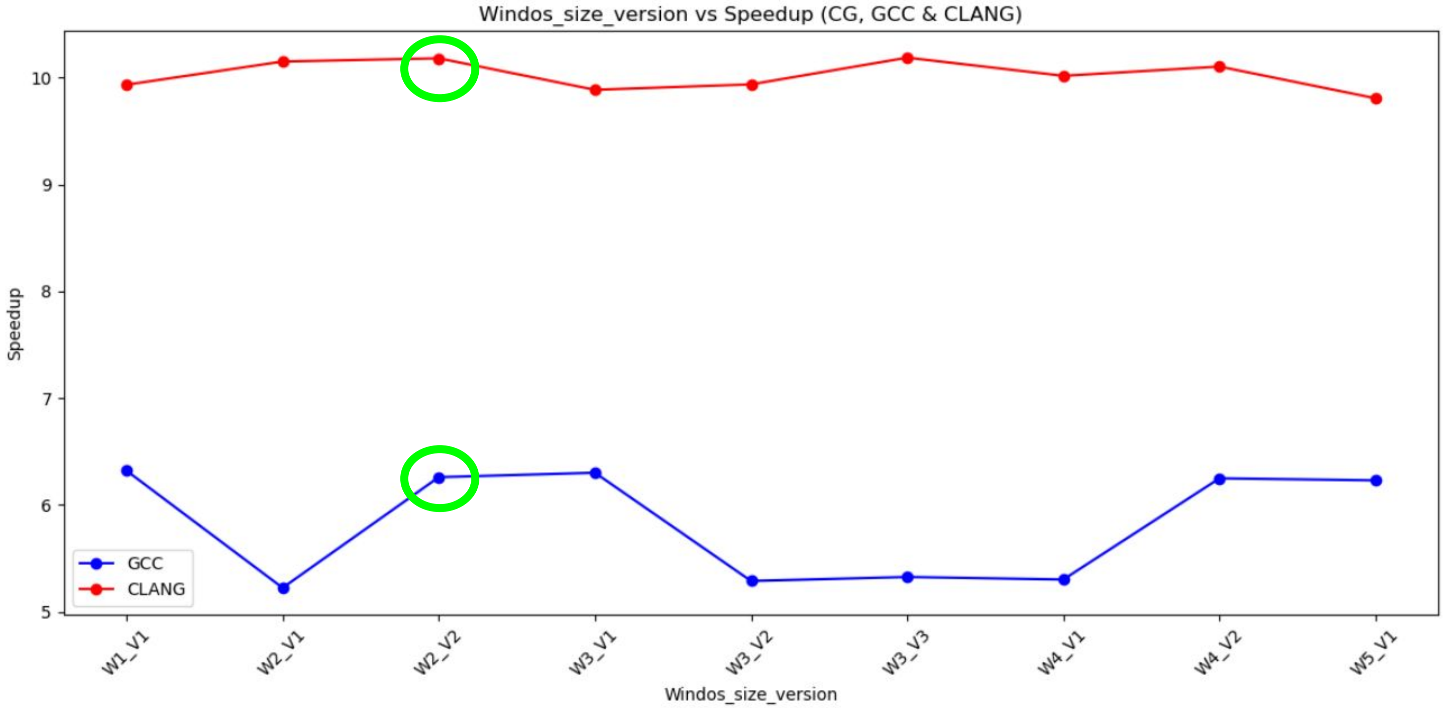
Windows Size Performance

Application CG from the Nas Parallel Benchmark suite. Subroutine conj_grad



Windows Size Performance

Application CG from the Nas Parallel Benchmark suite. Subroutine conj_grad



UNIVERSITY OF DELAWARE

Conclusions and Future Work

- The best Windows size depends on the program and the target application.
- Clang showed better performance in 5 out of the 6 applications evaluated
- GCC and Clang implement different optimization strategies, runtime libraries (especially for OpenMP)
- Tuning two optimizing compilers
- Using ML power in order to navigate the huge search space

Questions?