THE UNIVERSITY of EDINBURGH
informatics

EPSRC Centre for Doctoral Training in
Pervasive Parallelism

Augmenting Type Signatures for Program Synthesis

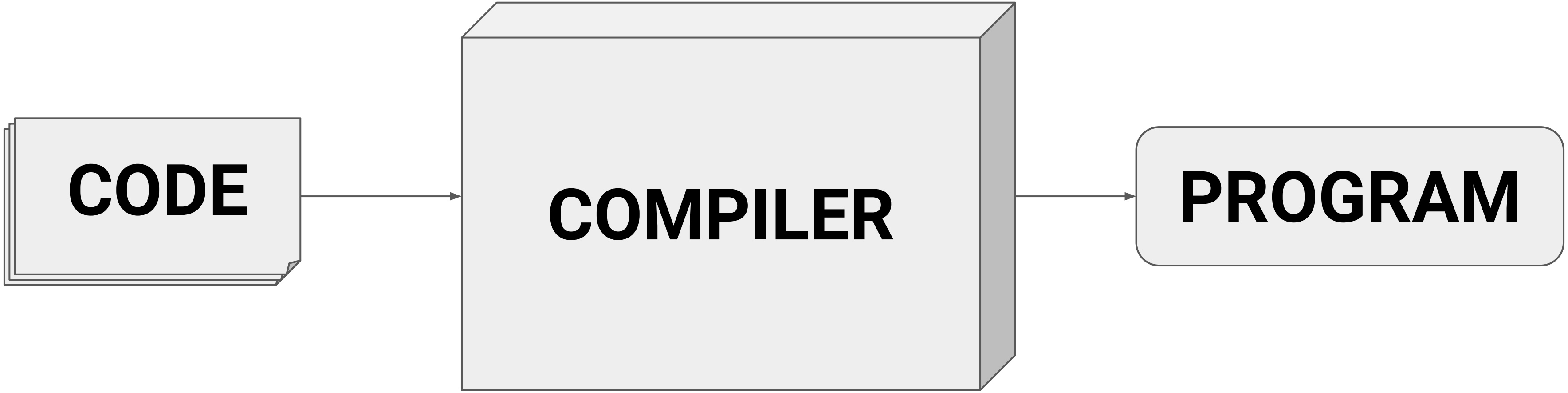
Bruce Collie
University of Edinburgh

bruce.collie@ed.ac.uk
baltoli.github.io

ICFP 19 BERLIN

TyDe 2019

A Compiler

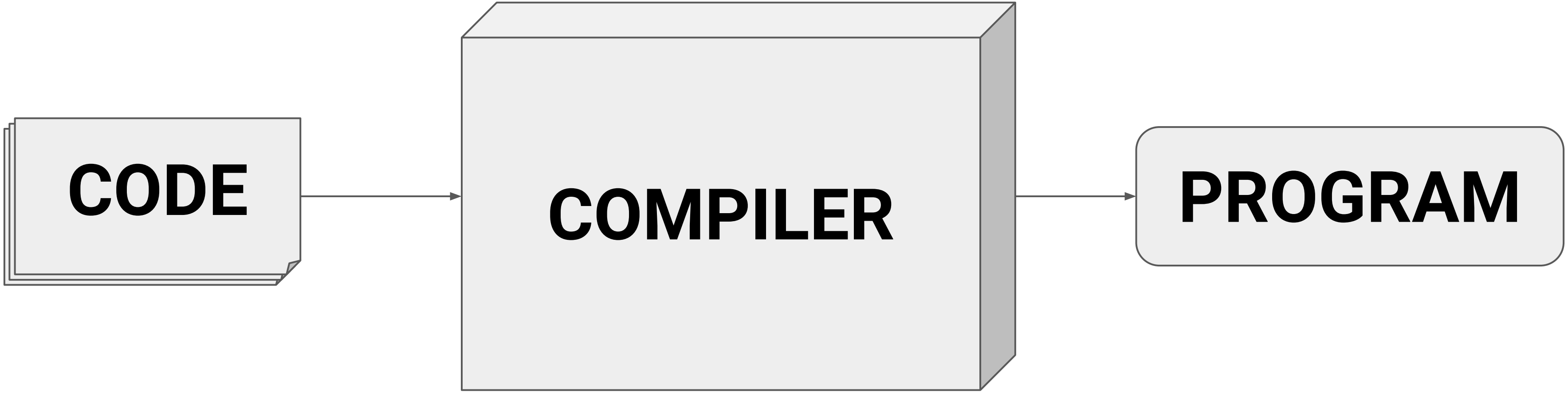


CODE

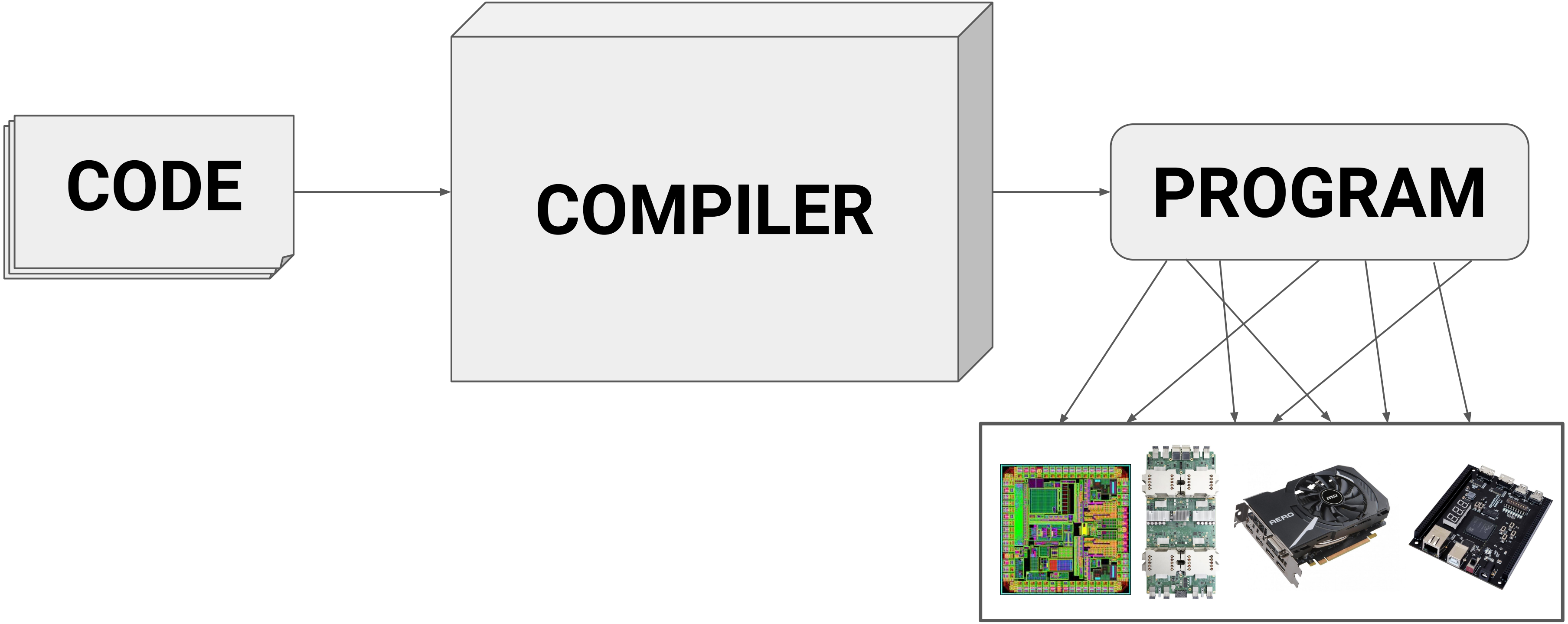
COMPILER

PROGRAM

“A Sufficiently Smart Compiler”



“A Sufficiently Smart Compiler”

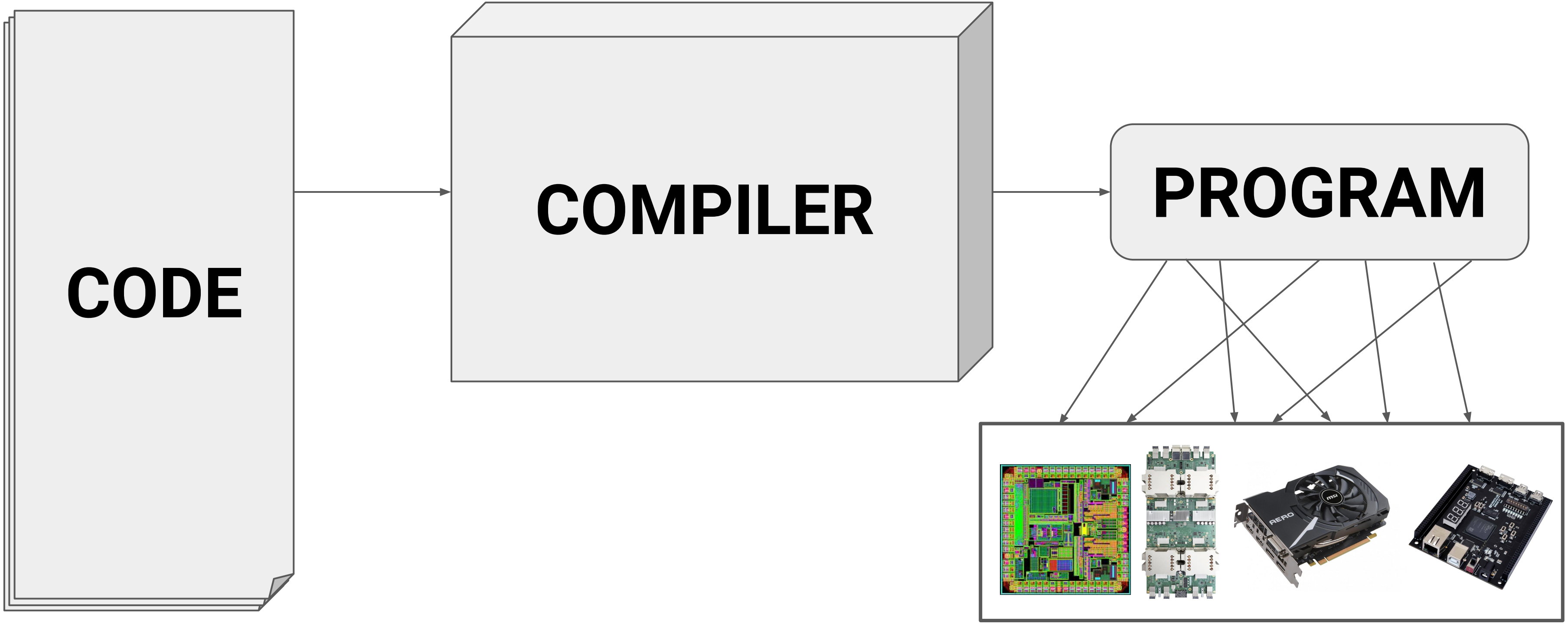


CODE

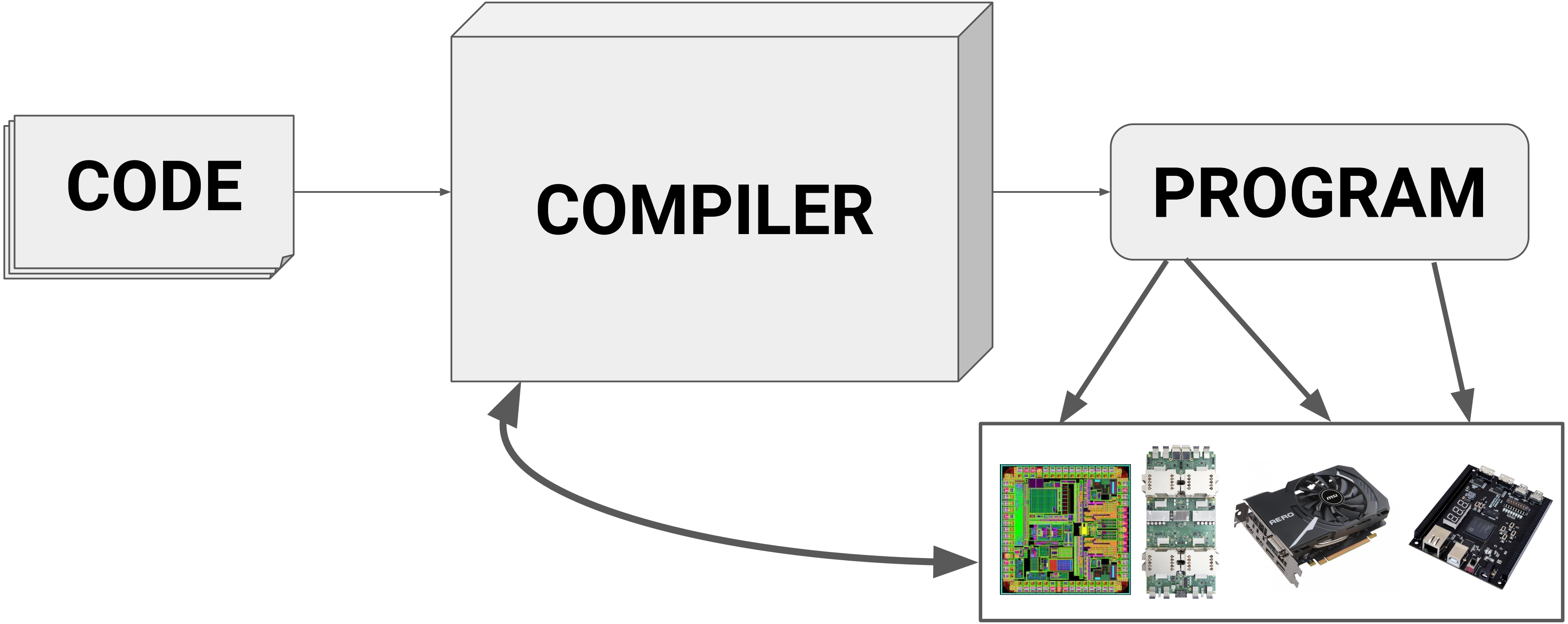
COMPILER

PROGRAM

“A Sufficiently Smart Compiler”



“A Sufficiently Smart Compiler”

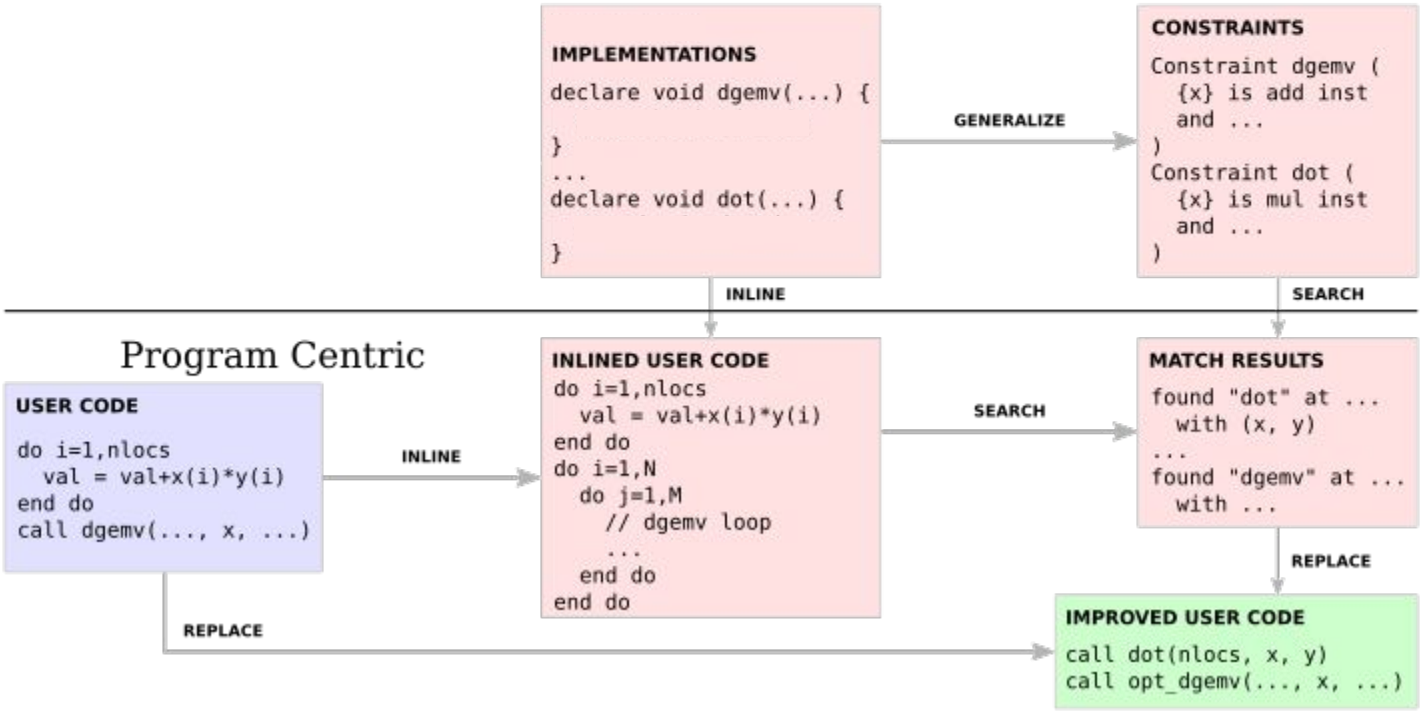


CODE

COMPILER

PROGRAM

The Problem



IMPLEMENTATIONS

```
declare void dgemv(...) {

}
...
declare void dot(...) {

}
```

CONSTRAINTS

```
Constraint dgemv (
  {x} is add inst
  and ...
)
Constraint dot (
  {x} is mul inst
  and ...
)
```

USER CODE

```
do i=1,nlocs
  val = val+x(i)*y(i)
end do
call dgemv(..., x, ...)
```

INLINED USER CODE

```
do i=1,nlocs
  val = val+x(i)*y(i)
end do
do i=1,N
  do j=1,M
    // dgemv loop
    ...
  end do
end do
```

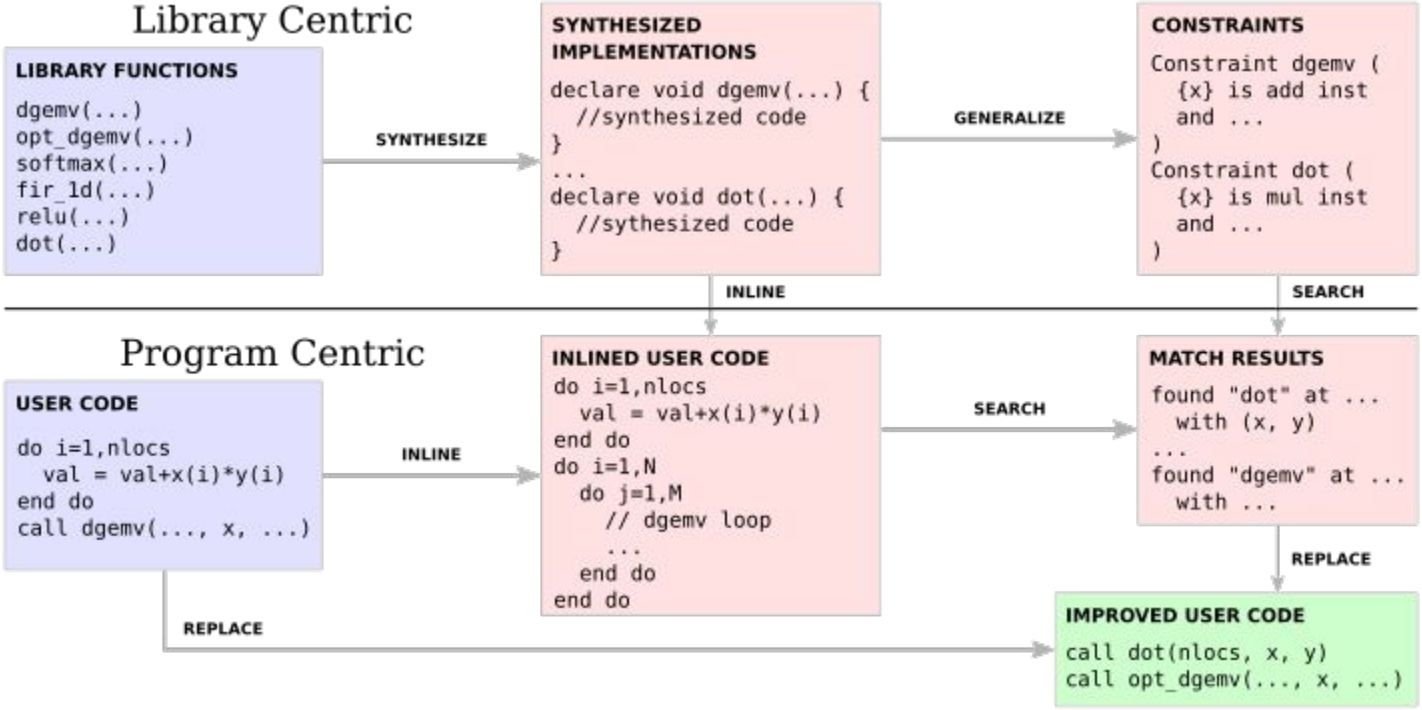
MATCH RESULTS

```
found "dot" at ...
  with (x, y)
...
found "dgemv" at ...
  with ...
```

IMPROVED USER CODE

```
call dot(nlocs, x, y)
call opt_dgemv(..., x, ...)
```

The Problem



Library Centric

LIBRARY FUNCTIONS

```
dgemv(...)
opt_dgemv(...)
softmax(...)
fir_1d(...)
relu(...)
dot(...)
```

SYNTHESIZED IMPLEMENTATIONS

```
declare void dgemv(...) {
  //synthesized code
}
...
declare void dot(...) {
  //sythesized code
}
```

CONSTRAINTS

```
Constraint dgemv (
  {x} is add inst
  and ...
)
Constraint dot (
  {x} is mul inst
  and ...
)
```

Program Centric

USER CODE

```
do i=1,nlocs
  val = val+x(i)*y(i)
end do
call dgemv(..., x, ...)
```

INLINED USER CODE

```
do i=1,nlocs
  val = val+x(i)*y(i)
end do
do i=1,N
  do j=1,M
    // dgemv loop
    ...
  end do
end do
```

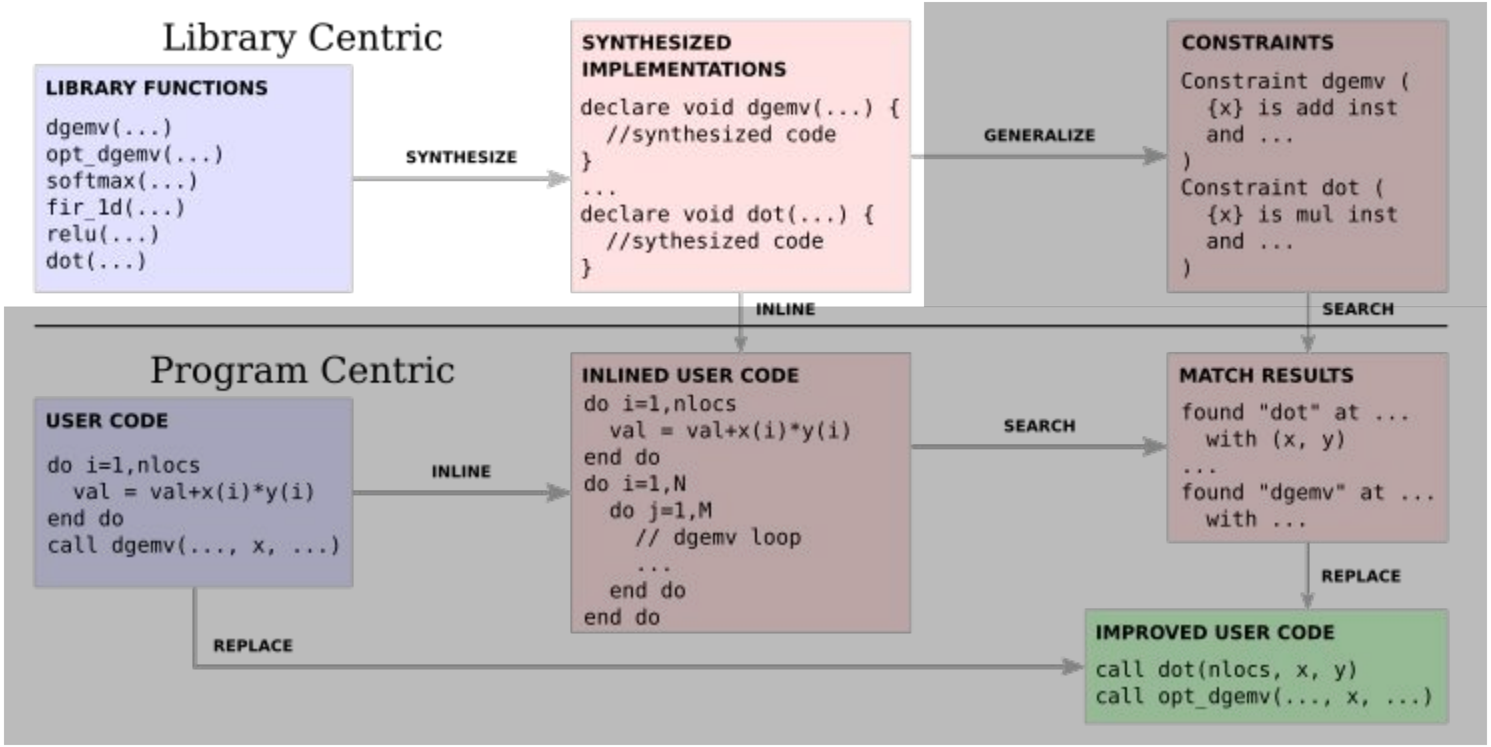
MATCH RESULTS

```
found "dot" at ...
  with (x, y)
...
found "dgemv" at ...
  with ...
```

IMPROVED USER CODE

```
call dot(nlocs, x, y)
call opt_dgemv(..., x, ...)
```

The Problem



LIBRARY FUNCTIONS

```
dgemv(...)
opt_dgemv(...)
softmax(...)
fir_1d(...)
relu(...)
dot(...)
```

SYNTHESIZED IMPLEMENTATIONS

```
declare void dgemv(...) {
  //synthesized code
}
...
declare void dot(...) {
  //sythesized code
}
```

CONSTRAINTS

```
Constraint dgemv (
  {x} is add inst
  and ...
)
Constraint dot (
  {x} is mul inst
  and ...
)
```

USER CODE

```
do i=1,nlocs
  val = val+x(i)*y(i)
end do
call dgemv(..., x, ...)
```

INLINED USER CODE

```
do i=1,nlocs
  val = val+x(i)*y(i)
end do
do i=1,N
  do j=1,M
    // dgemv loop
    ...
  end do
end do
```

MATCH RESULTS

```
found "dot" at ...
  with (x, y)
...
found "dgemv" at ...
  with ...
```

IMPROVED USER CODE

```
call dot(nlocs, x, y)
call opt_dgemv(..., x, ...)
```

The C Type System

```
char, int, float, double, void, ...

struct S { int x; float y; }

int *, char[], struct S*, ...

void (*f)(int, float*);
```

Example

```
void gemv(
  int m, int n,
  float *a, float *x, float *y)
{
  // ???
}
```

Example

```c
void gemv(
  int m, int n,
  float *a, float *x, float *y)
{
  for (int i = 0; i < m; ++i)
    for (int j = 0; j < n; ++j)
      y[i] += x[j] * a[j + i*n];
}
```

Example

```
void gemv(
  int m, int n,
  float *a, float *x, float *y)
{
  // ???
}
```

Non-Type Properties

```
void gemv(
  int m, int n,
  float *a, float *x, float *y);
```

Non-Type Properties

```
void gemv(
  int m, int n,
  float *a, float *x, float *y);
```

“y points to at least m elements”

“x points to at least n elements”

“y is an output”

Non-Type Properties

```
void gemv(
  int m, int n,
  float *a, float *x, float *y);
```

```
size(y, m)
size(x, n)
output(y)
```

Templates

```
for(x : xs) { [?] }

[?]; [?]

if(v == c) { [?] } else { [?] }

modify_compilation([?])

… etc.
```

Templates

```
for(x : xs) { [?] }

[?]; [?]

if(v == c) { [?] } else { [?] }

modify_compilation([?])

… etc.
```

COMPOSITIONAL

Templates

```
for(x : xs) { [?] }

[?]; [?]

if(v == c) { [?] } else { [?] }

modify_compilation([?])

… etc.
```

COMPOSITIONAL

LLVM IR RECIPES

Templates

```
for(x : xs) { [?] }

[?]; [?]

if(v == c) { [?] } else { [?] }

modify_compilation([?])

… etc.
```

COMPOSITIONAL

LLVM IR RECIPES

PARAMETERIZED

Queries

`P(A,B)`

PROPERTIES

Queries

`P(A,B)`	**PROPERTIES**
`P(A,B) and Q(B, C)`	**CONJUNCTION, UNIFICATION**

Queries

`P(A,B)`	**PROPERTIES**
`P(A,B) and Q(B, C)`	**CONJUNCTION, UNIFICATION**
`P(A, B) and no R(B, C)`	**NEGATION**

Queries

`P(A,B)`	**PROPERTIES**
`P(A,B) and Q(B, C)`	**CONJUNCTION, UNIFICATION**
`P(A, B) and no R(B, C)`	**NEGATION**

`Type(A, int)`
`Pointer(A)`
	**TYPE SIGNATURE,
	STANDARD
	QUERIES**

GEMV

```
void gemv(
    int m, int n,
    float *a, float *x, float *y);
```

```
size(y, m)
size(x, n)
 output(y)
```

GEMV

```
void gemv(
  int m, int n,
  float *a, float *x, float *y);
```

```
size(y, m)
size(x, n)
 output(y)
```

→

```
output(V)
  and Type(V, T)
  => Store(T, V)
```

GEMV

```
void gemv(
  int m, int n,
  float *a, float *x, float *y);
```

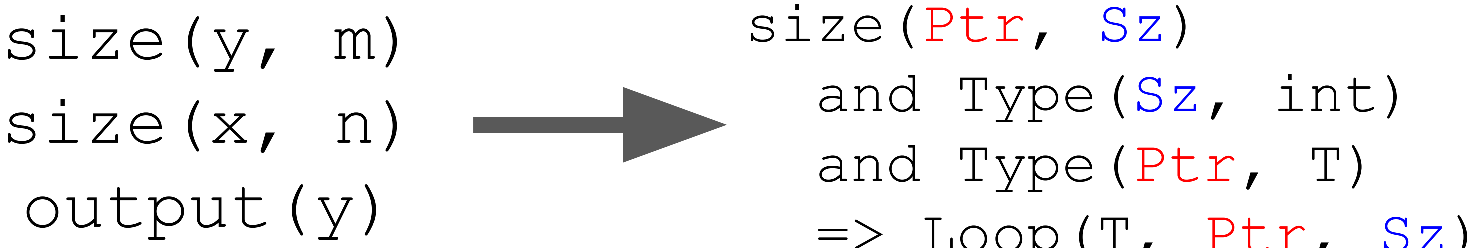
```
size(y, m)
size(x, n)
 output(y)
```

→

```
output(y)
  and Type(y, float*)
  => Store(float*, y)
```

GEMV

```
void gemv(
  int m, int n,
  float *a, float *x, float *y);
```



```
size(y, m)
size(x, n)
 output(y)
```

```
size(Ptr, Sz)
  and Type(Sz, int)
  and Type(Ptr, T)
  => Loop(T, Ptr, Sz)
```

GEMV

```
void gemv(
  int m, int n,
  float *a, float *x, float *y);
```

```
size(y, m)
size(x, n)
 output(y)
```

```
size(y, m)
  and Type(m, int)
  and Type(y, float*)
  => Loop(float*, y, m)
```

GEMV

```
void gemv(
   int m, int n,
   float *a, float *x, float *y);
```

```
size(y, m)
size(x, n)
 output(y)
```

→

```
size(x, n)
  and Type(n, int)
  and Type(x, float*)
  => Loop(float*, x, n)
```

GEMV

```
void gemv(
  int m, int n,
  float *a, float *x, float *y);
```

```
size(y, m)
size(x, n)
 output(y)
```

→

```
Pointer(Ptr)
  and Type(Ptr, T)
  and no size(Ptr, Sz)
  => computeIdx(T, Ptr)
```

GEMV

```
void gemv(
   int m, int n,
   float *a, float *x, float *y);
```

```
size(y, m)              Pointer(a)
size(x, n)     →          and Type(a, float*)
 output(y)                and no size(a, ?)
                          => computeIdx(float*, a)
```

GEMV

```
void gemv(
  int m, int n,
  float *a, float *x, float *y);
```

```
size(y, m)
size(x, n)
 output(y)
```

→

```
Loop(float*, y, m)
Loop(float*, x, n)
Store(float*, y)
ComputeIdx(float*, a)
```

GEMV

```c
void gemv(
  int m, int n,
  float *a, float *x, float *y)
{
  for (int i = 0; i < m; ++i)
    for (int j = 0; j < n; ++j)
      y[i] += x[j] * a[j + i*n];
}
```

GEMV

```
        void gemv(
          int m, int n,
          float *a, float *x, float *y)
        {
LOOP ───→ for (int i = 0; i < m; ++i)
            for (int j = 0; j < n; ++j)
              y[i] += x[j] * a[j + i*n];
        }
```

GEMV

```
       void gemv(
         int m, int n,
         float *a, float *x, float *y)
       {
LOOP →   for (int i = 0; i < m; ++i)
LOOP →     for (int j = 0; j < n; ++j)
             y[i] += x[j] * a[j + i*n];
       }
```

GEMV

```
      void gemv(
        int m, int n,
        float *a, float *x, float *y)
      {
LOOP ——→ for (int i = 0; i < m; ++i)
LOOP ———→  for (int j = 0; j < n; ++j)
             y[i] += x[j] * a[j + i*n];
      }
           STORE ↗
```

GEMV

```
void gemv(
  int m, int n,
  float *a, float *x, float *y)
{
  for (int i = 0; i < m; ++i)
    for (int j = 0; j < n; ++j)
      y[i] += x[j] * a[j + i*n];
}
```

LOOP

LOOP

STORE

INDEX

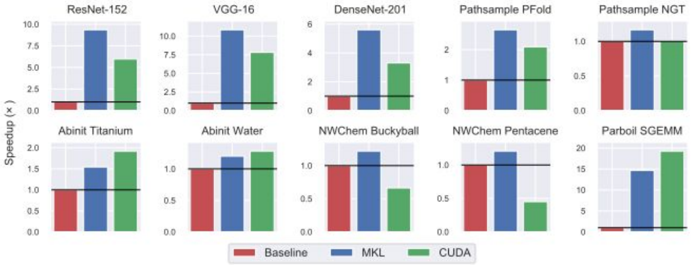
Results

PACT 2019

- Performance focus
- Linear algebra, scientific code, ML

Results

PACT 2019



Results

PACT 2019

Type-Directed Program Synthesis and Constraint Generation for Library Portability

Results

PACT 2019

Type-Directed Program Synthesis and Constraint Generation for Library Portability

IN PROGRESS

- **100+ functions, 7 libraries**
- **Generalization + integration**
- **Varied domains + use cases**