Euro LLVM 2018

16-17 April 2018

Parallware, LLVM & Supercomputing

Manuel Arenaz
manuel.arenaz@apppentra.com

appentra
Software for guided parallelization

Index

- **Motivation for Supercomputing, LLVM & Parallware**
- Parallware Software Architecture

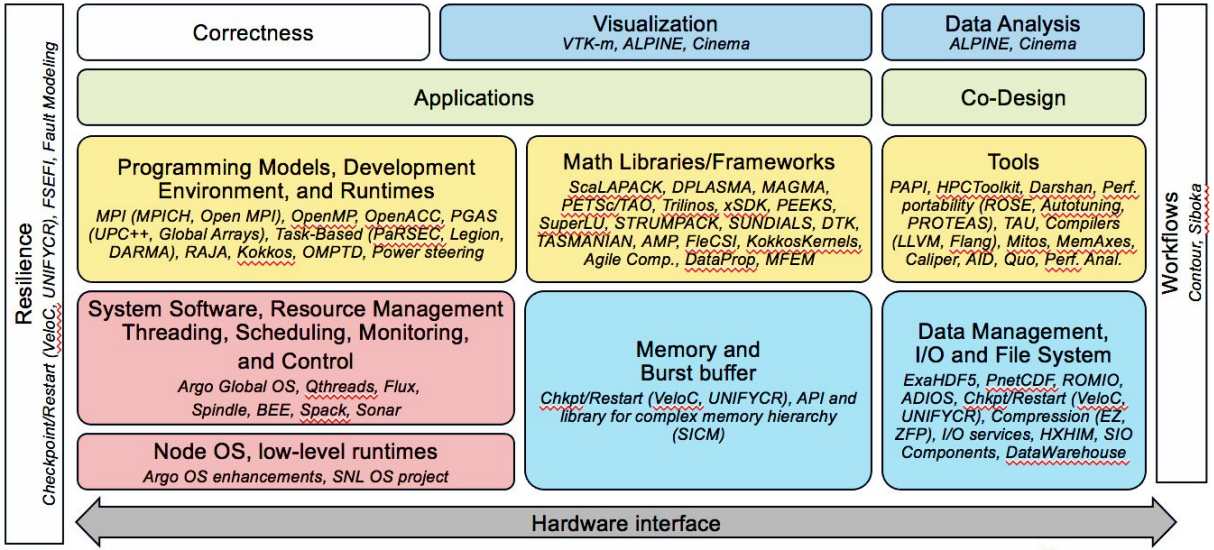
Why Supercomputing? (Top500 Nov 2017)

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
2	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
3	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 Cray Inc.	361,760	19,590.0	25,326.3	2,272
4	Japan Agency for Marine-Earth Science and Technology Japan	**Gyoukou** - ZettaScaler-2.2 HPC system, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz ExaScaler	19,860,000	19,135.8	28,192.0	1,350
5	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209

Supercomputers provide **massive parallelism** to address **leading-edge scientific problems**

Why LLVM?

CLANG FLANG



https://www.hpcwire.com/2017/04/26/messina-update-u-s-path-exascale-15-slides/

Parallel programming for **scientific codes** is based on a **complex software stack**

Why Parallware?

ATMUX
Multiplication of Transposed Sparse Matrix by Vector

Serial version		**gcc**	-O2 -march=native -mtune=native -ftree-vectorize	**0.25 s**	
		clang	-O2 -march=native -mtune=native -ftree-vectorize	**0.27 s**	
PARALLWARE manages race conditions with...	OpenMP pragma atomic	**gcc**	-O2 -march=native -mtune=native -ftree-vectorize **-fopenmp**	**0.359 s**	**0.7x**
		clang	-O2 -march=native -mtune=native -ftree-vectorize **-fopenmp**	**0.3470 s**	**0.78x**
	OpenMP privatization of arrays	**gcc**	-O2 -march=native -mtune=native -ftree-vectorize **-fopenmp**	**0.1172 s**	**2.13x**
		clang	-O2 -march=native -mtune=native -ftree-vectorize **-fopenmp**	**0.1177 s**	**2.29x**

Test system: Intel® Core™ i7-4770 CPU @ 3.40GHz

atmux.c

```
1  void atmux(double* val, double* x, double* y,
2             int* col_ind, int* row_ptr, int n, int nz)
3  {
4      for(int t = 0; t < n; t++)
5          y[t] = 0;
6
7      for(int i = 0; i < n; i++) {
8          for (int k = row_ptr[i]; k < row_ptr[i+1]; k++) {
9              y[col_ind[k]] = y[col_ind[k]] + x[i] * val[k];
10         }
11     }
12 }
```

Parallware helps to boost the performance of scientific codes

What is Parallware?

Parallelization of (sequential) source code with minimal user-provided compiler hints

atmux.c

```
1  void atmux(double* val, double* x, double* y,
2             int* col_ind, int* row_ptr, int n, int nz)
3  {
4      for(int t = 0; t < n; t++)
5          y[t] = 0;
6
7      for(int i = 0; i < n; i++) {
8          for (int k = row_ptr[i]; k < row_ptr[i+1]; k++) {
9              y[col_ind[k]] = y[col_ind[k]] + x[i] * val[k];
10         }
11     }
12 }
```

pw_atmux_private.c | pw_atmux.c | pw_struct_atmux.c | pw_struct_atmux_ACC.c | pw_struct_atmux_test.c

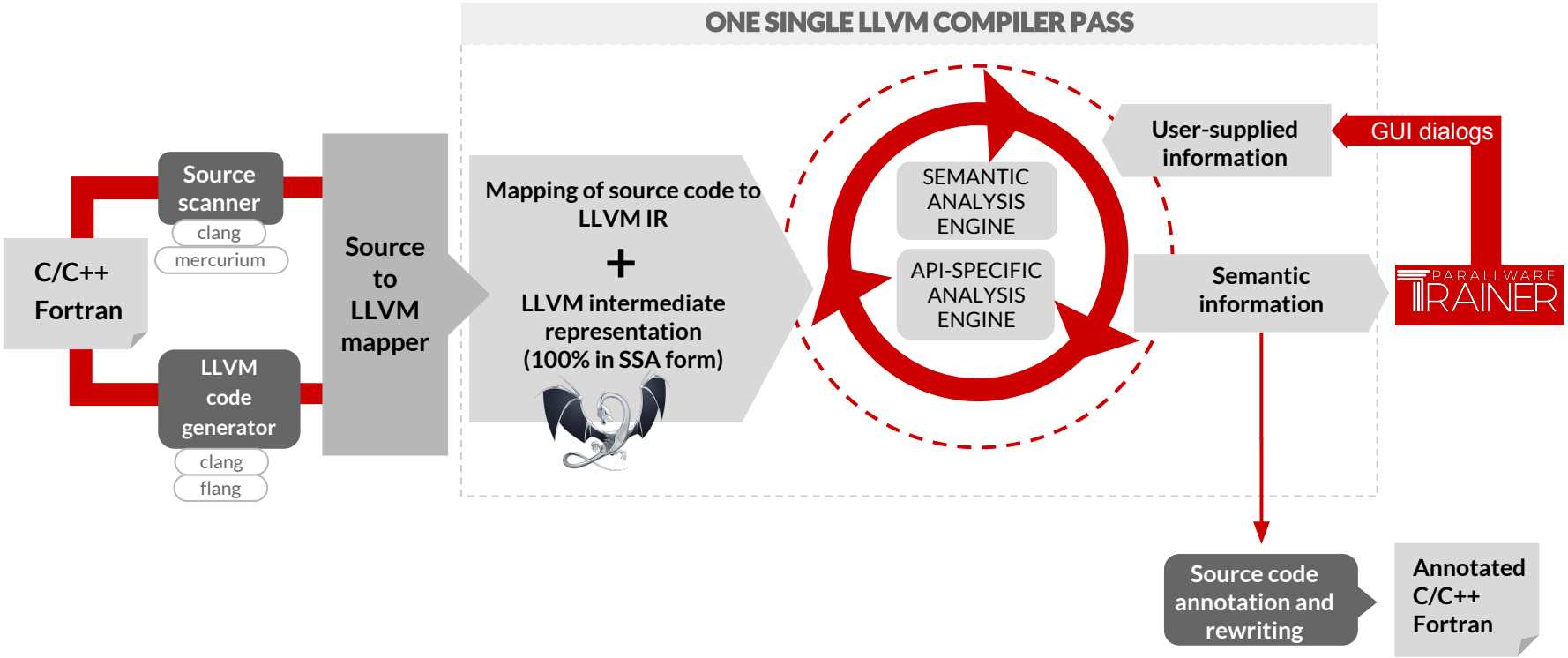
```
1  #include <stdlib.h>
2
3  void atmux(double* val, double* x, double* y,
4             int* col_ind, int* row_ptr, int n, int nz)
5  {
6      double *y_private;
7
8      #pragma omp parallel default(none) shared(col_ind, n, row_ptr, val, x, y) private(y_private)
9      {
10     #pragma omp for schedule(auto)
11     for(int t = 0; t < n; t++)
12         y[t] = 0;
13
14     y_private = (double*) malloc(sizeof(double)*n);
15     for(int pw0=0; pw0<n; ++pw0) {
16         y_private[pw0] = 0;
17     }
18     #pragma omp for schedule(auto)
19     for(int i = 0; i < n; i++) {
20         for (int k = row_ptr[i]; k < row_ptr[i+1]; k++) {
21             y_private[col_ind[k]] = y_private[col_ind[k]] + x[i] * val[k];
22         }
23     }
24     #pragma omp critical
25     {
26         for(int pw0=0; pw0<n; ++pw0) {
27             y[pw0] += y_private[pw0];
28         }
29     }
30     free(y_private);
31     } // end parallel
32 }
```

Parallware(R) is a static program analysis technology to discover parallelization opportunities

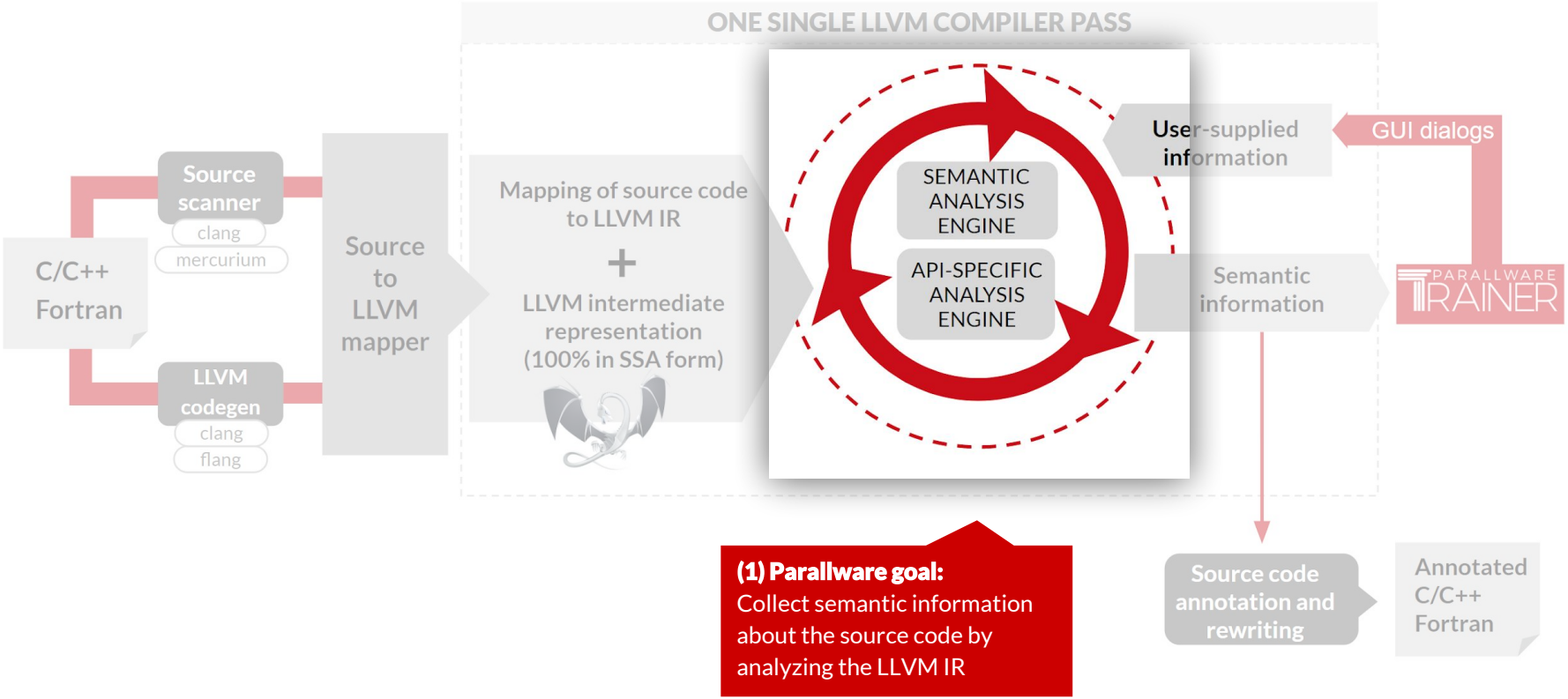
Index

- Motivation for Supercomputing, LLVM & Parallware
- **Parallware Software Architecture**

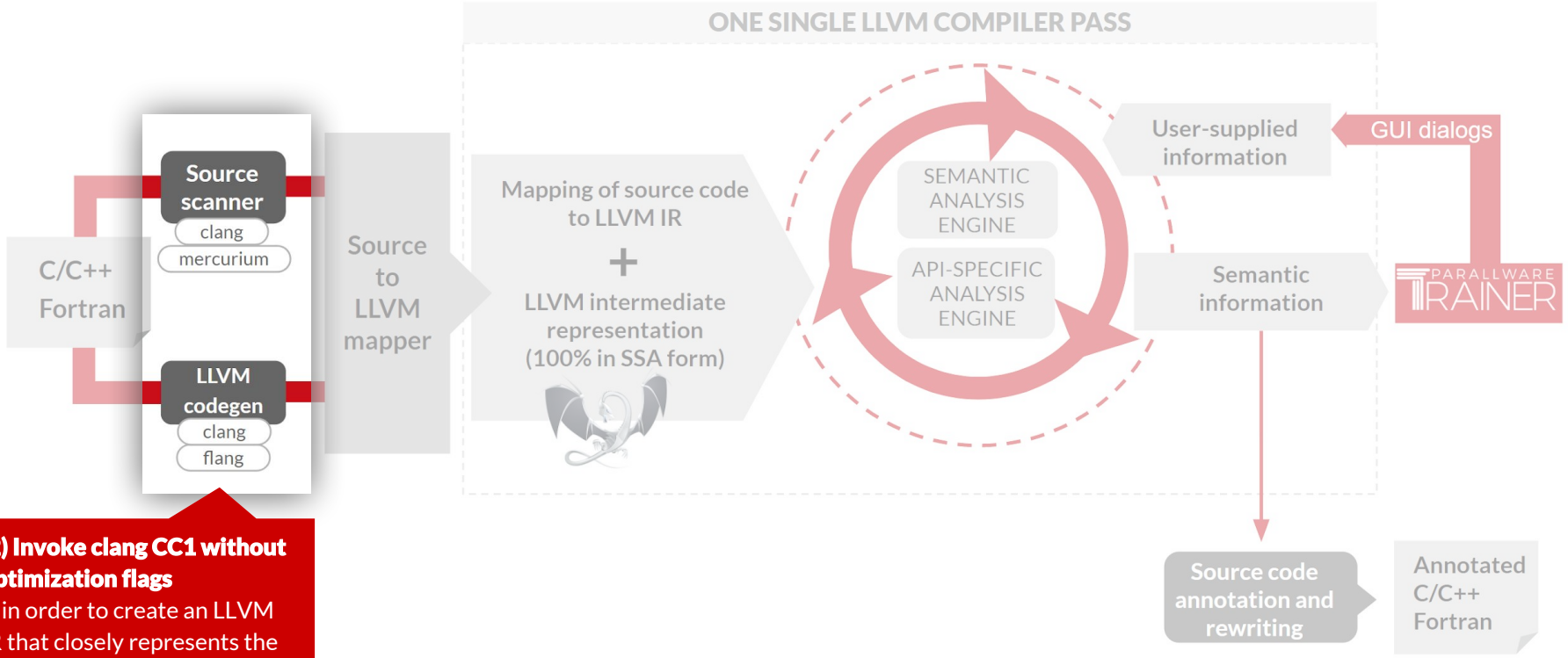
Parallware Software Architecture



Parallware Software Architecture

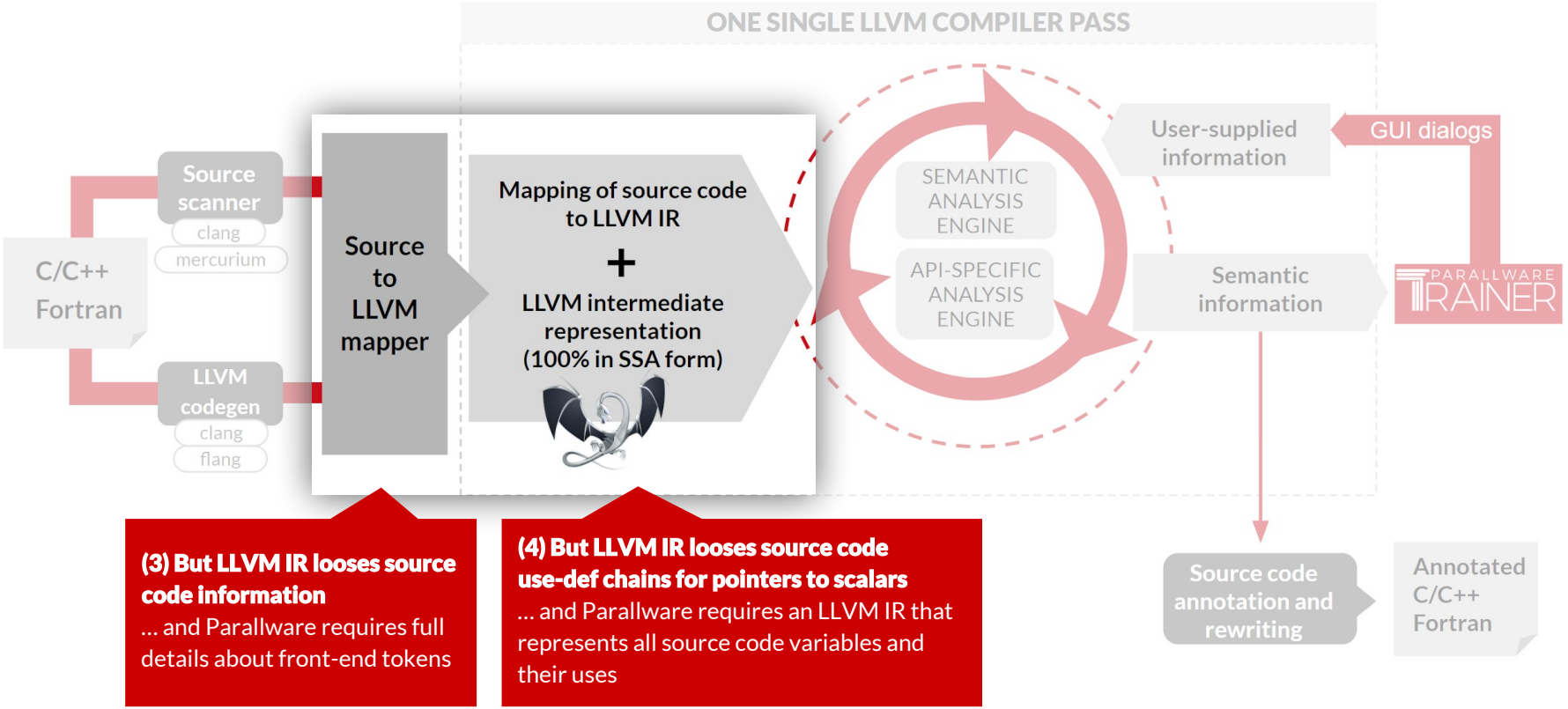


Parallware Software Architecture



(2) Invoke clang CC1 without optimization flags
… in order to create an LLVM IR that closely represents the original source code

Parallware Software Architecture



Mem2reg Value propagation

Challenge #1:

- Mem2reg removes the LLVM instructions AllocaInst/LoadInst/StoreInst of scalar variables converted into SSA form.

Impact:

- Parallware semantic engine needs to analyze the computations of all variables in order to generate OpenMP pragmas/clauses.
- Key information to determine the data scoping of variable (e.g., shared, private, reduction, firstprivate, lastprivate).

```
int f() {
    int x = 1;
    int y = x;
    int z = y + 2;
    return z;
}
```

```
define i32 @f() #0 {
        %1 = alloca i32, align 4
        %2 = alloca i32, align 4
        %3 = alloca i32, align 4
        store i32 1, i32* %1, align 4
        %4 = load i32, i32* %1, align 4
        store i32 %4, i32* %2, align 4
        %5 = load i32, i32* %2, align 4
        %6 = add nsw i32 %5, 2
        store i32 %6, i32* %3, align 4
        %7 = load i32, i32* %3, align 4
        ret i32 %7
}
```

```
define i32 @f() local_unnamed_addr #0 !dbg !7 {
        %1 = add nsw i32 1, 2
        ret i32 %1
}
```

LLVM/Clang ABI Implementation

Challenge #2:

- Source code data types are not always preserved by clang CC1
- Clang CC1 changes procedure's signature to be conformant with the target Application Binary Interface (ABI)

Impact:

- In order to generate OpenMP pragmas/clauses, Parallware needs to determine the signature of all the source code procedures.

```
struct S {
      int f[4];
};

void f(struct S s) {
      s.f[2] = 3;
      return;
}
```

```
%struct.S = type { [4 x i32] }

define void @f(i64, i64) #0 {
      %3 = alloca %struct.S, align 4
      %4 = bitcast %struct.S* %3 to { i64, i64 }*
      %5 = getelementptr inbounds { i64, i64 }, { i64, i64 }* %4, i32 0, i32 0
      store i64 %0, i64* %5, align 4
      %6 = getelementptr inbounds { i64, i64 }, { i64, i64 }* %4, i32 0, i32 1
      store i64 %1, i64* %6, align 4
      %7 = getelementptr inbounds %struct.S, %struct.S* %3, i32 0, i32 0
      %8 = getelementptr inbounds [4 x i32], [4 x i32]* %7, i64 0, i64 2
      store i32 3, i32* %8, align 4
      ret void
}
```

Short-circuit evaluation complexity

Challenge #3:

- Clang CC1 expands expressions as a set of BasicBlocks that evaluate the expression according to the C/C++ standard.
- Examples: short-circuit evaluation, ternary operator

Impact:

- In order to perform data flow analysis efficiently in large source codes, Parallware reduces the complexity of the Control Flow Graph of the LLVM IR.

```
int f(int x, int y) {
    if ( x && y ) {
        return 2;
    } else {
        return 3;
    }
}
```

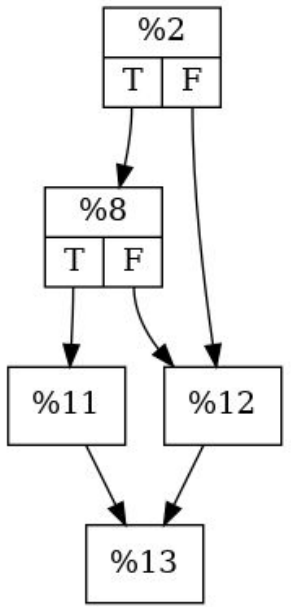
```
define i32 @f(i32, i32) #0 {
        %6 = load i32, i32* %4, align 4
        %7 = icmp ne i32 %6, 0
        br i1 %7, label %8, label %12

; <label>:8:                                     ; preds = %2
        %9 = load i32, i32* %5, align 4
        %10 = icmp ne i32 %9, 0
        br i1 %10, label %11, label %12

; <label>:11:                                    ; preds = %8
        store i32 2, i32* %3, align 4
        br label %13

; <label>:12:                                    ; preds = %8, %2
        store i32 3, i32* %3, align 4
        br label %13

; <label>:13:                                    ; preds = %12, %11
        %14 = load i32, i32* %3, align 4
        ret i32 %14
}
```



Euro LLVM 2018

16-17 April 2018

Parallware, LLVM & Supercomputing

Manuel Arenaz
manuel.arenaz@apppentra.com

appentra
Software for guided parallelization

LLVM/Clang ABI Implementation

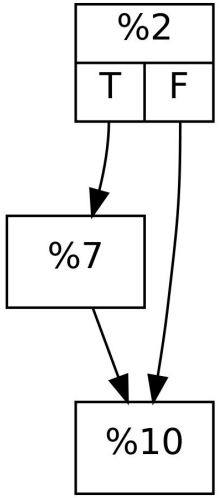
```
struct S {
      int f[16];
};

struct S f() {
      struct S s;
      s.f[2] = 3;
      return s;
}
```

```
define void @f(%struct.S* noalias sret) #0 {
      %2 = alloca %struct.S, align 4
      %3 = getelementptr inbounds %struct.S, %struct.S* %2, i32 0, i32 0
      %4 = getelementptr inbounds [16 x i32], [16 x i32]* %3, i64 0, i64 2
      store i32 3, i32* %4, align 4
      %5 = bitcast %struct.S* %0 to i8*
      %6 = bitcast %struct.S* %2 to i8*
      call void @llvm.memcpy.p0i8.p0i8.i64(i8* %5, i8* %6, i64 64, i32 4, i1 false)
      ret void
}
```

Short-circuit evaluation complexity

```
int f(int x, int y)
{
      return x && y;
}
```




```
define i32 @f(i32, i32) #0 {
      %3 = alloca i32, align 4
      %4 = alloca i32, align 4
      store i32 %0, i32* %3, align 4
      store i32 %1, i32* %4, align 4
      %5 = load i32, i32* %3, align 4
      %6 = icmp ne i32 %5, 0
      br i1 %6, label %7, label %10

; <label>:7:                                     ; preds = %2
      %8 = load i32, i32* %4, align 4
      %9 = icmp ne i32 %8, 0
      br label %10

; <label>:10:                                    ; preds = %7, %2
      %11 = phi i1 [ false, %2 ], [ %9, %7 ]
      %12 = zext i1 %11 to i32
      ret i32 %12
}
```

Short-circuit evaluation complexity

```
int g(int x);

int f(int x, int y) {
    return x ? g(x) : g(y);
}
```

```
define i32 @f(i32, i32) #0 {
      %3 = alloca i32, align 4
      %4 = alloca i32, align 4
      store i32 %0, i32* %3, align 4
      store i32 %1, i32* %4, align 4
      %5 = load i32, i32* %3, align 4
      %6 = icmp ne i32 %5, 0
      br i1 %6, label %7, label %10

; <label>:7:                                      ; preds = %2
      %8 = load i32, i32* %3, align 4
      %9 = call i32 @g(i32 %8)
      br label %13

; <label>:10:                                     ; preds = %2
      %11 = load i32, i32* %4, align 4
      %12 = call i32 @g(i32 %11)
      br label %13

; <label>:13:                                     ; preds = %10, %7
      %14 = phi i32 [ %9, %7 ], [ %12, %10 ]
      ret i32 %14
}
```