Introduction

EnergyPlus C++

Stuart G. Mentzer
Objexx Engineering, Inc.

What We Did: Round 1

EnergyPlus 8.0 Fortran was prepped/tested

- Vela Energy-Plus-Fortran repo

EnergyPlus 8.0 was converted to C++

- ObjexxFCL: arrays, strings, and intrinsics

Validated 8.0 (Thanks Edwin!)

* Issues found and fixed/noted in Fortran and C++

What We Did: Round 2

EnergyPlus 8.2 Fortran was prepped

- Merged into Energy-Plus-Fortran code

EnergyPlus 8.2 converted, merged, polished

Validated/debugged 8.2 (Edwin!)

* New issues reported: Some fixed, some not yet

Why We Did It

Opens code to wider developer pool

C++ Standard Library enables better code

Rich set of libs: Boost, Blitz++, Loki, Qt, …

Enables OOD to manage growing complexity

C++ is powerful and expressive (but not easy)

Converting to C++ is just the first step

Fortran Limitations as Apps Grow

Profusion of function arguments

Logic is distributed throughout code

Big, nested IF blocks for type discrimination

Arrays used for every data structure

Difficult to modify code without introducing bugs

Poor ecosystem of testing tools

What C++/OO Can Bring

Enables migration to a robust OOD

- Solid components with clean interfaces
  => Focus on the engineering
- Express subtle behaviors and relationships
- Good OO design is modular and extensible

C++ In a Nutshell

Type-safe with full OO support:

- Encapsulation/containment layering
- Dynamic polymorphism via inheritance
- Compile-time polymorphism via templates

Powerful but easy to get into trouble:

- Raw pointers and C-style arrays => overflows, leaks, ...
- Class hierarchy implementation subtleties: overrides, overloads, protected constructors, ...

Not Why We Did It

C++ is simpler and easier to learn than Fortran

C++ programs run faster than Fortran

C++ code is automatically object-oriented

C++ compiles and links quickly

C++ compilers produce clear error messages

Good News / Bad News

☺ Still looks a lot like the Fortran

- Easy to start working with
- Type safety and debug assertions reduce bugs
- Some bugs were exposed during conversion

☹ Still looks a lot like the Fortran

- Not object-oriented
- Conversion to C++ is an enabling step

What We Have

Fortran-like C++ very close to original

Not object-oriented (but TYPE to struct)

Mostly solid C++ but some Fortran-isms

- Post-conversion migrations are suggested

Code Remains Familiar

Fortran

```
DO A=1,LEN_TRIM(InputString)
  B=INDEX(UpperCase,InputString(A:A))
  IF (B .NE. 0) THEN
    OutputString(A:A)=LowerCase(B:B)
  ELSE
    OutputString(A:A)=InputString(A:A)
  END IF
END DO
```

C++

```
for ( A = 1; A <= len_trim( InputString ); ++A ) {
   B = index( UpperCase, InputString( A, A ) );
   if ( B != 0 ) {
     OutputString( A, A ) = LowerCase( B, B );
   } else {
     OutputString( A, A ) = InputString( A, A );
   }
}
```

Fortran to C++ Mappings

MODULE	namespace
TYPE	struct + constructors
SUBROUTINE sub(...)	void sub(...)
IF (cond) THEN	if (cond) {
DO i = 1, N	for (i = 1; i <= N; ++i)
WRITE(unit,fmt) a, b	gio::write(unit, fmt) << a << b;

Function Declarations

Fortran

```
REAL(r64) FUNCTION foo(i,x,o)
  INTEGER, INTENT(IN) :: i
  REAL(r64) :: x
  LOGICAL, OPTIONAL :: o
  …
END FUNCTION foo
```

C++

```
Real64
foo(
    int const i,
    Real64 & x,
    Optional_bool o
)
{…}
```

User-Defined Types

Fortran

```
TYPE :: Vector
  REAL(r64) :: x
  REAL(r64) :: y
  REAL(r64) :: z
END TYPE
```

Fortran generates constructors and assignment operators automatically

C++

```
struct Vector
{
    // Members
    Real64 x;
    Real64 y;
    Real64 z;

    // Default Constructor
    Vector()
    {}

    // Member Constructor
    Vector(
        Real64 const x,
        Real64 const y,
        Real64 const z
    ) :
        x( x ),
        y( y ),
        z( z )
    {}
};
```

Performance

Tuning needed to get back to Fortran speed

Typically focus on hot spot loops:

- Linear array indexing (like Fortran compiler)
- Common expression hoisting
- Algorithm refinements (FindArrayIndex 2+x speedup)
- Heap use and temporaries (ANY(A==B) is wasteful)

Future: OO and performance are not enemies

Getting both requires care

EnergyPlus Performance

EnergyPlus hot spots are *not* array/loop ops

- Lookups in nested arrays/objects in arrays/objects in …
- Conditional tests for state and type
- Simple expressions and assignments

⇨ Harder to tune but doable

Performance-limiting:

- Not vectorizable
- Cache unfriendly

Performance Tuning Case Study

RefrigeratedWarehouse example (annual run)

- Raw C++ more than 2x slower
- FindArrayIndex was top: New C++ is >2x faster
- With addl. tuning gprof time now matches Fortran
- C++ system lib time kept it 50% slower overall
- A lot of this was heap use that Fortran avoids
- Quick pass at some easy heap waste cut this by 40%

Potential Longer Range Goals

Migrate code to be more robust and natural C++

OO to simplify: modularity, extensibility

Increase focus on testing and testability

Exploit libraries and expressive power of C++

Attack flow to reach very high performance

Where Are We Going?

Near Term:

- Post-conversion migrations
- Structural improvements
- Simple OO migration

Evolutionary migration to OO architecture

Modern design: testable, no global data, ...

Refactoring for high performance

Questions