

# Phase Componentization in Open64

- recipe for adding optimization or analysis phase



Min Zhao  
06/09/2011

# Open64 Background

## Important Attributes

- In production since 1996, open-sourced in 2000 via GPL by SGI
- Most modern design among today's production compilers
- Infrastructure designed to facilitate optimization implementation
- Compatible and inter-operable with gcc/g++
- Widely used today in optimization research
- Easy to retarget to new processors



# Motivation for the Work

## Quotes from Open64 Researchers and Developers

- “The mechanisms for passing internal options to backend is inconsistent...”
- “The method of accessing an option differs from phase to phase”
- “... can a *new* PhD student easily plug in a new analysis without having to touch too many different modules in open64?”
- “... does open64 has modular design like LLVM?”
- “... Is it easy to develop a pass and plug into the tool chain?”



# Phase Componentization

## Objectives – a *simple* recipe for adding optimization/analysis phase

- Componentizing optimization/analysis phases in Open64
- Consistently Handling
  - options (enable, disable ...)
  - traces (print the trace information)
  - dumps (dump ir/ir2c/cfg/ssa/vcg before and after the phase)
  - statistics (time, memory statistics)
  - triaging mechanism (skip, limit, ...)
- Building and accessing the shared data structures easily (e.g. CFG, SSA)
- Reducing the ramp-up overhead for new developers



# Overview Design

## O64\_Component

### component common data

- Enable/disable
- Statistics
- Trace
- Local memory pool
- Invalidate shared data struct

## O64\_Driver

### shared data struct

- Component descriptor list
- Option array ([component][option])
- Current Whirl CFG
- Current Whirl SSA
- Current WN
- Memory pool



# Component Base and Driver Class

```
O64_Component:: O64_Component(component) {  
    _CurrentComponent = component;  
    _Driver = O64_Driver::GetInstance();  
    _CurrentOption = _Driver->GetCurrentOption();  
// handling skip option  
    if (Current_PU_Count() < _CurrentOption->  
        >GetUIntOption(_CurrentComponent, OPT_skip_b) || ... ) {  
        _enable = false; _disable = true;  
        return;  
    }  
    ... // setting other common data  
    ProcessDumpOptions_(); // handling dump before  
    StartEndMessage_(); // handling driver trace  
    ... // handling time statistics  
}
```

```
O64_Driver:: O64_Drvier() {  
    // Initialize the memory pool  
    MEM_POOL_Initialize(&_DriverPool, "DriverPool", FALSE);  
    MEM_POOL_Initialize(&_LocalPool, "LocalPool", FALSE);  
    MEM_POOL_Push(&_DriverPool);  
// Allocate for memory for all component  
descriptors  
}  
O64_Driver:: RegisterComponent(comp, desc) {  
    _ComponentDescriptorList.ComponentDescriptors[comp]=  
        const_cast<O64_ComponentDescriptor*>(desc);  
}  
O64_Driver::ProcessComponentOption(argv) {  
    // create the option array and set to default  
    // process argv and set specific option values  
}  
O64_Driver is a singleton;
```



# Adding a New Optimization Phase



Adding a new component name to the enum

O64\_COMPONENT

Creating a component class, which is derived from class

O64\_Component

Adding the component's public interface into header file and invoking component

Passing the options to the backend

# Adding a New Optimization Phase

## Step 1

### (component name)

Adding a new  
component name to  
the enum

O64\_COMPONENT  
in common/util/flags.h

## Example

```
typedef enum
{
    COMPONENT_invalid = -1,
    COMPONENT_first  = 0,
    COMPONENT_driver = COMPONENT_first,
    ...,
    COMPONENT_my_comp,
    COMPONENT_last

} O64_COMPONENT;
```



# Adding a New Optimization Phase

## Example

```
class OPT_MyComponent : public O64_Component
{
    public:
        // declare component option enum
        enum OPTION {
            OPT_mycomp_first = OPT_component_first,
            ...
            OPT_mycomp_last
        };
        // component & option descriptors are static members
        static const O64_ComponentDescriptor ComponentDescriptor;
        static const O64_OptionDescriptor OptionDescriptors[OPT_mycomp_last - OPT_mycomp_first + 1];
        // constructor and destructor
    private:
        // local data and functions
        Perform_(); // does the real work
};
```

## Step2

### (component class)

### Class skeleton



# Adding a New Optimization Phase

## Step2 (continued)

option descriptor,  
component descriptor,  
initializer: to register the  
component into  
O64\_Driver class

### Example

```
const O64_OptionDescriptor
OPT_MyComponent::OptionDescriptors[] = {
    O64_OPTION_DESC(OPT_mycomp_first, "first option", "OPT1", "opt1", OVK_BOOL,
        OV_INTERNAL, false, false, 0, 0),
    ...
    O64_OPTION_DESC(OPT_mycomp_last, "end marker", 0, 0, OVK_INVALID,
        OV_INTERNAL, false, 0, 0, 0)
}
const O64_ComponentDescriptor
OPT_MyComponent::ComponentDescriptor = {
    O64_COMPONENT_DESC("My component description", "MYCOMP",
        OptionDescriptors)
};
static O64_ComponentInitializer mycomp_init(
    COMPONENT_my_comp, &OPT_MyComponent::ComponentDescriptor);
```



# Adding a New Optimization Phase

## Example

```
OPT_MyComponent::OPT_MyComponent() : O64_Component(COMPARTMENT_my_comp)
{
    // enable by default (OR disable by default)
    if (!_disable) return;

    // you may want to cache option values in private fields like
    _Option1 = _CurrentOptions->GetBoolOption(_CurrentComponent, OPT_mycomp_first);

    // get the shared data structure from O64_Driver if needed
    _CurrentWN = _Driver->GetCurrentWN();
    _CurrentWCFG = _Driver->GetCurrentWCFG();
    _CurrentWSSA = _Driver->GetCurrentWSSA();

    // call to do the real work
    Perform_();
}
```

## Step2 (continued)

Constructor:  
to prepare the data and  
call to do the real work



# Adding a New Optimization Phase

## Step3

### (Component Invocation)

Component entry point  
declaration in

common/com/comp\_decl.h ;

Component entry point;

Component invocation;

## Example

```
// entry point declaration in comp_decl.h  
extern void OPT_PerformMyComponent();
```

```
// entry point should be defined in component file  
void OPT_PerformMyComponent() {  
    OPT_MyComponent opt_mycomp();  
}
```

```
// invoking the optimization at a appropriate place in the driver  
// for example inside Pre_Optimizer  
Pre_Optimizer() {  
    ...  
    OPT_PerformMyComponent();  
    ...  
}
```



# Adding a New Optimization Phase

## Step4

### (Option support)

Passing the options of a component to the backend by adding an entry in driver/OPTIONS

## Example

```
-DRIVER:%s ; ALL be "-DRIVER:%s"  
"options for internal use in driver"
```

## New component option format

```
-COMP:[option_name=option_value+]*
```

For example: -DRIVER:trace+options+stat=time



# What a component has...

- Unified option handling

- Common options:

-COMP:options, enable, disable, trace=info[min/max], db=ir[ir2c/cfg/ssa/vcg]

- Component specific options: -COMP:option=value

- Easy access of shared data structures (e.g. CFG, SSA)

- O64\_Driver::GetCurrentWCFG();

- Useful driver trace and statistics

- -DRIVER:trace,



# Summary

## Takeaways

- Adding a new optimization/analysis phase becomes easy
- Allow consistent handling of traces, dumps, options, statistics, triaging
- Help the new developers come-up to speed quickly

**“Everything should be made as simple as possible, but not simpler.”**

**- Albert Einstein**



# Thank You!

