Architecture Specific Code Generation and Function Multiversioning

Eric Christopher (echristo@gmail.com)

Talk Outline

Motivation

Current Status and Changes

Future Work

Motivation

Where are we coming from?

Link Time Optimization and Architecture Interworking

Function Multiversioning

Subtarget Architecture Support

X86: SSE3, SSSE3, SSE4.2, AVX

ARM: NEON, ARM, Thumb

Mips: Mips32, Mips16, Mips3d

PowerPC: VSX

Subtarget Interworking

```
static inline __attribute__((mips16)) int i1 (void) { return 1; }
static inline __attribute__((nomips16)) int i2 (void) { return 2; }
static inline __attribute__((mips16)) int i3 (void) { return 3; }
static inline __attribute__((nomips16)) int i4 (void) { return 4; }

int __attribute__((nomips16)) f1 (void) { return i1 (); }
int __attribute__((mips16)) f2 (void) { return i2 (); }
int __attribute__((mips16)) f3 (void) { return i3 (); }
int __attribute__((nomips16)) f4 (void) { return i4 (); }
```

Subtarget LTO

```
clang -g -c foo.c -emit-llvm -o foo.bc -mavx2

clang -g -c bar.c -emit-llvm -o bar.bc

clang -g -c baz.c -emit-llvm -o baz.bc -mavx2

llvm-link foo.bc bar.bc baz.bc -o lto.bc

clang lto.bc -o lto.x
```

foo.c:

```
int foo_avx(void *x, int a) {
  return _mm_aeskeygenassist_si128(x, a);
}
```

bar.c:

```
int foo_generic(void *x, int a) {
  // Lots of code
}
```

baz.c:

```
const unsigned AVXBits = (1 << 27) | (1 << 28);
  bool HasAVX = ((ECX & AVXBits) == AVXBits) && OSHasAVXSupport();
  bool HasAVX2 = HasAVX && MaxLeaf >= 0x7 &&
                 !GetX86CpuIDAndInfoEx(0x7, 0x0, &EAX, &EBX, &ECX, &EDX) &&
                 (EBX & 0x20);
  GetX86CpuIDAndInfo(0x80000001, &EAX, &EBX, &ECX, &EDX);
 if (HasAVX)
  return foo_avx(x, a);
 else
  return foo_generic(x, a);
```

Function Multiversioning

Avoid splitting code between files.

Avoid expensive runtime checks.

Performance and code size benefits of per-cpu features.

```
__attribute__ ((target ("default")))
int foo () {
  // The default version of foo.
  return 0;
}
__attribute__((target("sse4.2"))) int foo() {
  // foo version for SSE4.2
  return 1;
}
__attribute__((target("arch=atom"))) int foo() {
  // foo version for the Intel ATOM processor
  return 2;
}
int main() {
  int (*p)() = &foo;
  assert((*p)() == foo());
  return 0;
}
```

Function Multiversioning - Linux/IFUNC

Functions are specially mangled

All calls go through the PLT

Dispatch function is generated to determine CPU features

Special symbol type and relocation to help minimize the dispatch overhead

Why not a function pointer?

Another function to do the dispatch one call through the PLT
for a shared library

Then the indirect call through the function table

With IFUNC the PLT resolves to the method that gets chosen by the IFUNC resolver

```
define float @_Z3barv() #0 {
entry:
  ret float 4.000000e+00
}
define float @_Z4testv() #1 {
entry:
  ret float 1.000000e+00
}
define float @_Z3foov() #2 {
entry:
  ret float 4.000000e+00
}
define float @_Z3bazv() #3 {
entry:
  ret float 4.000000e+00
}
attributes #0 = { "target-cpu"="x86-64" "target-features"="+avx2" }
attributes #1 = { "target-cpu"="x86-64" }
attributes #2 = { "target-cpu"="corei7" "target-features"="+sse4.2" }
attributes #3 = { "target-cpu"="x86-64" "target-features"="+avx2" }
```

Talk Outline

Motivation

Current Status and Changes

Future Work

Subtarget Specific Code Generation

```
TargetSubtargetInfo &ST =   const_cast<TargetSubtargetInfo&>(TM.
getSubtarget<TargetSubtargetInfo>());
ST.resetSubtargetFeatures(MF);
```

Only works for instruction selection

Requires a global lock on the Subtarget - no parallel code generation!

```
define float @_Z3barv() #0 {
entry:
  ret float 4.000000e+00
}
define float @_Z4testv() #1 {
entry:
  ret float 1.000000e+00
}
define float @_Z3foov() #2 {
entry:
  ret float 4.000000e+00
}
define float @_Z3bazv() #3 {
entry:
  ret float 4.000000e+00
}
attributes #0 = { "target-cpu"="x86-64" "target-features"="+avx2" }
attributes #1 = { "target-cpu"="x86-64" }
attributes #2 = { "target-cpu"="corei7" "target-features"="+sse4.2" }
attributes #3 = { "target-cpu"="x86-64" "target-features"="+avx2" }
```

TargetMachine

Target Options

Subtarget

Data Layout

Instruction Selection Information

Frame Lowering

Scheduling

Pass Manager

Object File Layout

TargetMachine

Target Options

Subtarget 

Data Layout

Instruction Selection Information

Frame Lowering

Scheduling

Pass Manager

Object File Layout

TargetMachine

Target Options

Subtarget 

Data Layout

Instruction Selection Information 

Frame Lowering

Scheduling

Pass Manager

Object File Layout

TargetMachine

Target Options

Subtarget 

Data Layout

Instruction Selection Information 

Frame Lowering 

Scheduling

Pass Manager

Object File Layout

TargetMachine

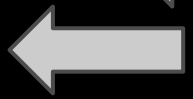
Target Options

Subtarget 

Data Layout

Instruction Selection Information 

Frame Lowering 

Scheduling 

Pass Manager

Object File Layout

TargetMachine

Target Options

Data Layout

Pass Manager

Object File Layout

Handles everything for the Target and Object File emission.

Subtarget Cache

getSubtarget still exists

```
template <typename STC> const STC &getSubtarget(const Function *) const

mutable StringMap<std::unique_ptr<STC>> SubtargetMap
```

```
const X86Subtarget *X86TargetMachine::getSubtargetImpl(const Function &F) const {
  AttributeSet FnAttrs = F.getAttributes();
  Attribute CPUAttr =
      FnAttrs.getAttribute(AttributeSet::FunctionIndex, "target-cpu");
  Attribute FSAttr =
      FnAttrs.getAttribute(AttributeSet::FunctionIndex, "target-features");

  std::string CPU = !CPUAttr.hasAttribute(Attribute::None)
                        ? CPUAttr.getValueAsString().str()
                        : TargetCPU;
  std::string FS = !FSAttr.hasAttribute(Attribute::None)
                       ? FSAttr.getValueAsString().str()
                       : TargetFS;

  auto &I = SubtargetMap[CPU + FS];
  if (!I) {
    resetTargetOptions(F);
    I = llvm::make_unique<X86Subtarget>(TargetTriple, CPU, FS, *this,
                                        Options.StackAlignmentOverride);
  }
  return I.get();
}
```

```
const X86Subtarget *X86TargetMachine::getSubtargetImpl(const Function &F) const {
  AttributeSet FnAttrs = F.getAttributes();
  Attribute CPUAttr =
      FnAttrs.getAttribute(AttributeSet::FunctionIndex, "target-cpu");
  Attribute FSAttr =
      FnAttrs.getAttribute(AttributeSet::FunctionIndex, "target-features");

  std::string CPU = !CPUAttr.hasAttribute(Attribute::None)
                        ? CPUAttr.getValueAsString().str()
                        : TargetCPU;
  std::string FS = !FSAttr.hasAttribute(Attribute::None)
                       ? FSAttr.getValueAsString().str()
                       : TargetFS;

  auto &I = SubtargetMap[CPU + FS];
  if (!I) {
    resetTargetOptions(F);
    I = llvm::make_unique<X86Subtarget>(TargetTriple, CPU, FS, *this,
                                        Options.StackAlignmentOverride);
  }
  return I.get();
}
```

Subtarget Cache

Implemented for X86, ARM, AArch64, Mips

Trivial to implement for other architectures

TargetTransformInfo

Uses a lot of Subtarget specific information

Pass manager doesn’t support boundary crossing analysis passes

So we need a function specific TTI

```
class FunctionTargetTransformInfo final : public FunctionPass {
private:
  const Function *Fn;
  const TargetTransformInfo *TTI;
public:
  void
  getUnrollingPreferences(Loop *L,
                          TargetTransformInfo::UnrollingPreferences
&UP) const {
    TTI->getUnrollingPreferences(Fn, L, UP);
  }
};
```

Talk Outline

Motivation

Current Status and Changes

Future Work

IR Changes

Function attribute cpu and feature string

```
attributes #0 = { "target-cpu"="x86-64" "target-features"="+avx2" }
```

New call/invoke destination for IFUNC calls

Optimization Directions

CFG Cloning

Auto-Autovectorization

Advanced Idiom Recognition

Questions?