Noise: User-Defined Optimization Strategies

Ralf Karrenberg, Marcel Köster, Roland Leißa,
Yevgeniya Kovalenko, Sebastian Hack

karrenberg@cs.uni-saarland.de

European LLVM Conference, Paris
April 30, 2013

Optimizing Legacy HPC Code

```
float bar(float x) { return x + 42.f; }

void foo(float x, float* in, float* out, int N) {


    for (int i=0; i<N; ++i) {
      float lic = x * bar(x);
      out[i] = in[i] + lic;
    }
    for (int i=0; i<N; ++i) {
      out[i] *= x;
    }

}
```

- “-O3” often yields undesired code

Optimizing Legacy HPC Code

```
float bar(float x) { return x + 42.f; }

void foo(float x, float* in, float* out, int N) {


    for (int i=0; i<N; ++i) {
      float lic = x * bar(x);
      out[i] = in[i] + lic;
    }
    for (int i=0; i<N; ++i) {
      out[i] *= x;
    }

}
```

- "-O3" often yields undesired code
- Option 1: Rewrite code manually

Optimizing Legacy HPC Code: Manual Rewriting

```
void foo(float x, float* in, float* out, int N) {
  float  lic  = x * (x + 42.f);
  __m256 licV = _mm256_set1_ps(lic);
  __m256 xV   = _mm256_set1_ps(x);
  int i = 0;
  if (N >= 32)
  {
  for ( ; i<N; i+=32) {
    __m256 in0 = _mm256_load_ps(in[i]);
    __m256 in1 = _mm256_load_ps(in[i+8]);
    __m256 in2 = _mm256_load_ps(in[i+16]);
    __m256 in3 = _mm256_load_ps(in[i+32]);
    _mm256_store_ps(out[i],    _mm256_mul_ps(_mm256_add_ps(in0, licV), xV));
    _mm256_store_ps(out[i+8],  _mm256_mul_ps(_mm256_add_ps(in1, licV), xV));
    _mm256_store_ps(out[i+16], _mm256_mul_ps(_mm256_add_ps(in2, licV), xV));
    _mm256_store_ps(out[i+32], _mm256_mul_ps(_mm256_add_ps(in3, licV), xV));
  }
  }
  for ( ; i<N; ++i) {
    out[i] = (in[i] + lic) * x;
  }
}
```

Optimizing Legacy HPC Code

```
float bar(float x) { return x + 42.f; }

void foo(float x, float* in, float* out, int N) {


    for (int i=0; i<N; ++i) {
      float lic = x * bar(x);
      out[i] = in[i] + lic;
    }
    for (int i=0; i<N; ++i) {
      out[i] *= x;
    }

}
```

- "-O3" often yields undesired code
- Option 1: Rewrite code manually

Optimizing Legacy HPC Code

```
float bar(float x) { return x + 42.f; }

void foo(float x, float* in, float* out, int N) {
  NOISE("loop-fusion inline(bar) licm vectorize(8) unroll(4)")
  {
    for (int i=0; i<N; ++i) {
      float lic = x * bar(x);
      out[i] = in[i] + lic;
    }
    for (int i=0; i<N; ++i) {
      out[i] *= x;
    }
  }
}
```

- “-O3” often yields undesired code
- Option 1: Rewrite code manually
- Option 2: Noise: Define what phases to run on a code segment

Specialized Loop Dispatching

```
// Before:
NOISE("specialize(a=1,2,3)")
{
    for (int i = 0; i < a; ++i) { ... }
}

// After:
switch(a)
{
case 1:
    for (int i = 0; i < 1; ++i) { ... }
    break;
case 2:
    for (int i = 0; i < 2; ++i) { ... }
    break;
case 3:
    for (int i = 0; i < 3; ++i) { ... }
    break;
default:
    for (int i = 0; i < a; ++i) { ... }
    break;
}
```

Conclusion

- Noise: Create user-defined optimization strategies
- Tune code without rewriting it
- Special-purpose transformations: loop dispatching, vectorization, . . .
- Minimally invasive extension to Clang
- Reintegration into Clang trunk?
- Prototype is being evaluated at HLRS Stuttgart
- Open source soon: `www.cdl.uni-saarland.de/projects/noise`

Conclusion

- Noise: Create user-defined optimization strategies
- Tune code without rewriting it
- Special-purpose transformations: loop dispatching, vectorization, . . .
- Minimally invasive extension to Clang
- Reintegration into Clang trunk?
- Prototype is being evaluated at HLRS Stuttgart
- Open source soon: `www.cdl.uni-saarland.de/projects/noise`

Thank You!

Visit us during the poster session ☺