Simple Outer Loop Vectorization == Loop Unroll-and-Jam + SLP

Dibyendu Das
AMD

Simple OLV == Loop Unroll-And-Jam (UnJ) + SLP

- OLV can be visualized as [Nuzman & Zaks, PACT 2008]
  - Unroll the outer loop by k times
  - Jam all the k-inner loop instances of the outer loop
  - Vectorize the loops using SLP
- Ex:

Original

```
for ( i = 0; i < N; i++ ) {
    accum = 0;
    for ( j = 0; j < 5; j++)
         accum += in[j][i] * filter[j];
    out[i] = sqrtf(accum)/particles;
}
```

After Unroll

```
for ( i = 0; i < N; i+= 4 ) {
    accum1 = accum2 =
        accum3 = accum4 = 0;
    for ( j = 0; j < 5; j++)
        accum1 += in[j][i] * filter[j];
    for ( j = 0; j < 5; j++)
        accum2 += in[j][i+1] * filter[j];
    for ( j = 0; j < 5; j++)
        accum3 += in[j][i+2] * filter[j];
    for ( j = 0; j < 5; j++)
        accum4 += in[j][i+3] * filter[j];
    out[i] = sqrtf(accum1)/particles;
    out[i+1] = sqrtf(accum2)/particles;
    out[i+2] = sqrtf(accum3)/particles;
    out[i+3] = sqrtf(accum4)/particles;

}
```

After UnJ

```
for ( i = 0; i < N; i+= 4 ) {
    accum1 = accum2 = … 0;
    for ( j = 0; j < 5; j++) {
         accum1 += in[j][i] * filter[j];
         accum2 += in[j][i+1] * filter[j];
         accum3 += in[j][i+2] * filter[j];
         accum4 += in[j][i+3] * filter[j];
    }
    out[i] = sqrtf(accum1)/particles;
    out[i+1] = sqrtf(accum2)/particles;
    out[i+2] = sqrtf(accum3)/particles;
    out[i+3] = sqrtf(accum4)/particles;

}
```

After SLP

```
<v_particles> = bcast<vparticles>
for ( i = 0; i < N; i+= 4 ) {
    <v_accum> = bcast<0,…,0>;
    for ( j = 0; j < 5; j++) {
         <v_accum> +=
              ld <in[j][i],…,in[j][i+3]> * bcast<filter[j]>;
    }
   st <out[i],…,out[i+3]> =
            vsqrtf(<v_accum>)/<v_particles>;
}
```

Better code generation of inner loop reduction
No gather in the inner loop

Loop Unroll-And-Jam

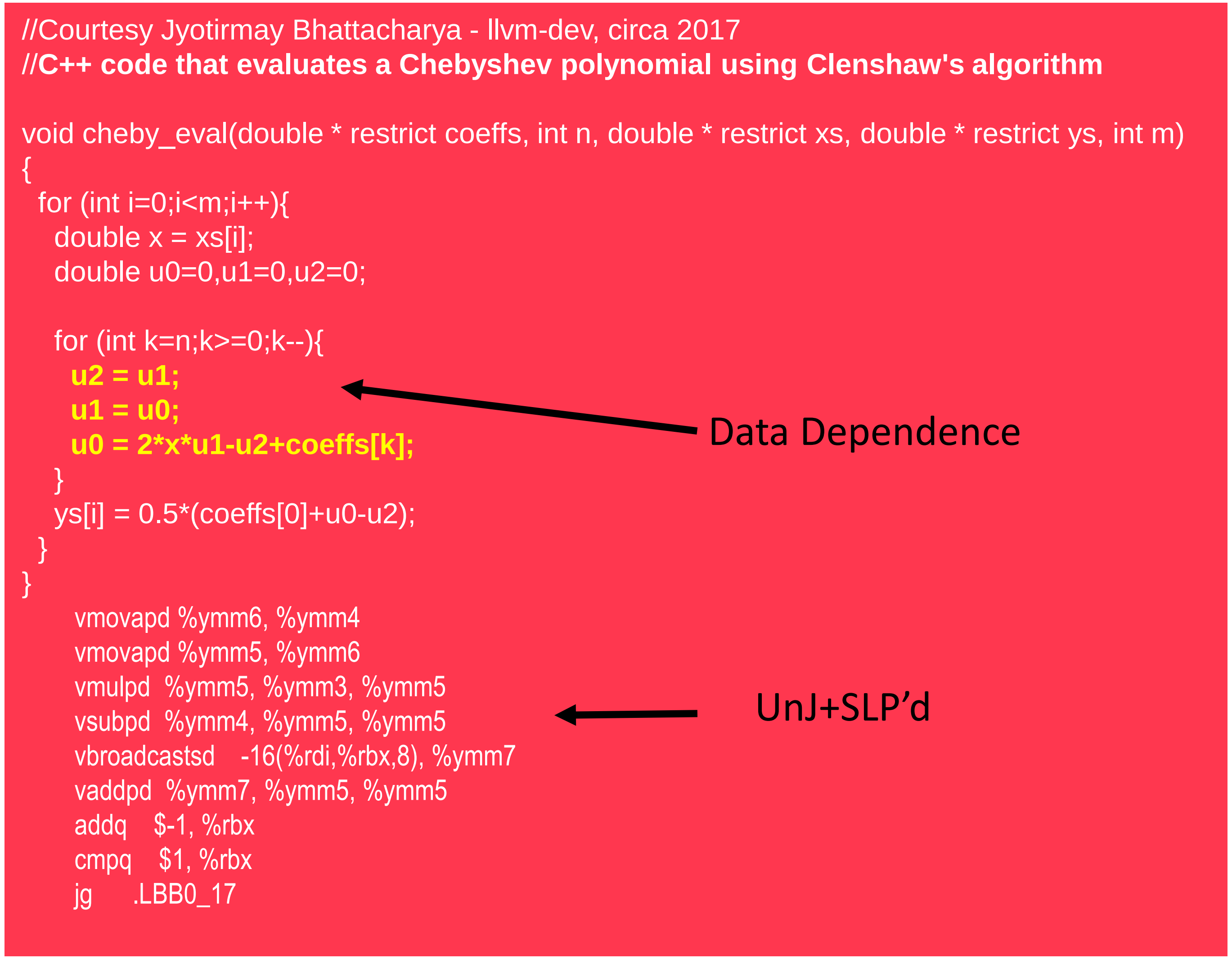
- New Pass introduced in July 2018
  - lib/Transforms/Scalar/LoopUnrollAndJamPass.cpp
- Two flags *–enable-unroll-and-jam* and *–allow-unroll-and-jam*
- Supports pragma *allow_unroll_and_jam(factor)*
- Called ‘after’ SLP in PassManager
  - Scheduling UnJ after SLP is late for our purpose

Modifications in IPO/PassManager to support OLV

- Schedule UnJ Pass before the LoopVectorizer Pass
- Call a bunch of cleanup routines after that
  - Looks like we may need to call LSR as a cleanup pretty early (challenging ?)
  - LSR needed probably because UnJ implementation is not optimal
- … ⇨ UnJ ⇨ *cleanup* ⇨ LV ⇨ … ⇨ SLP ⇨ …
- Need to schedule SLP also before LV ?
  - … ⇨ UnJ ⇨ *cleanup* ⇨ SLP’ ⇨ … ⇨ LV ⇨ … ⇨ SLP’ ⇨ …
  - Else LV may vectorize the jammed inner loop resulting in code which we don’t like ?
  - Very likely that due to costing LV will not vectorize the inner loop
    - Even if it does, we can modify SLP to SLP’ to vectorize “already-vectorized” code

One more example

- Reported in llvm-dev in 2017
  - Inner loop data dependence
  - No outer loop simdization pragma
    - Expects automatic OLV
- UnJ+SLP does OLV
  - Current llvm stage does some OLV but not cleanly
  - mul, sub not vectorized



```
//Courtesy Jyotirmay Bhattacharya - llvm-dev, circa 2017
//C++ code that evaluates a Chebyshev polynomial using Clenshaw's algorithm

void cheby_eval(double * restrict coeffs, int n, double * restrict xs, double * restrict ys, int m)
{
  for (int i=0;i<m;i++){
    double x = xs[i];
    double u0=0,u1=0,u2=0;

    for (int k=n;k>=0;k--){
      u2 = u1;
      u1 = u0;
      u0 = 2*x*u1-u2+coeffs[k];
    }
    ys[i] = 0.5*(coeffs[0]+u0-u2);
  }
}
      vmovapd %ymm6, %ymm4
      vmovapd %ymm5, %ymm6
      vmulpd  %ymm5, %ymm3, %ymm5
      vsubpd  %ymm4, %ymm5, %ymm5
      vbroadcastsd    -16(%rdi,%rbx,8), %ymm7
      vaddpd  %ymm7, %ymm5, %ymm5
      addq    $-1, %rbx
      cmpq    $1, %rbx
      jg      .LBB0_17
```

Data Dependence

UnJ+SLP’d

Open Problems

- Costing and Feasibility
  - Which loops to UnJ
    - Inner loops with reductions
    - Inner Loops with accesses strided on the outer loop index
    - Inner Loops with low trip count
    - Inner loops with data dependence but no dependence on the outer loops
  - What is the unroll factor (UF) ?
    - Assume SLP will work in which case choose UF such that DataSize * UF = SIMD width

DISCLAIMER AND ATTRIBUTIONS

DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18