

Project Proposal:

From our research, Clang's heuristic for deciding when to perform vectorization uses hand-picked costs assigned to each individual instruction. This instruction cost based heuristic correlates poorly with actual performance gains [1], and more sophisticated machine learning heuristics have been shown to outperform it in loop vectorization [2]. Our team proposes extending the work in NeuroVectorizer [2] by implementing a machine learning heuristic for SLP that examines the context of neighboring instructions in a basic block. Finally, we will measure the performance of our heuristic empirically by evaluating it against clang's individual instruction cost heuristic by compiling a testsuite of benchmarks and comparing their runtime results.

Proof of Concept:

Clang's cost model for SLP outperforms GCC in figure 1, however it misses the opportunity to perform SLP in figure 2. We suspect this is because Clang estimates SLP cost at an instruction level instead of using a heuristic that examines the context of the whole basic block.

```
1 #define N 1024
2 #define M 32
3 #define K 4
4
5 #define ALIGNED16 __attribute__((aligned(16)))
6
7 int a[M*2] ALIGNED16;
8 int b[M*2] ALIGNED16;
9 int c[M*2] ALIGNED16;
10 int d[M*2] ALIGNED16;
11
12 __attribute__((noinline))
13 void example1(int x1, int x2, int x3, int x4) {
14     a[0] = x1;
15     a[1] = x2;
16     a[2] = x3;
17     a[3] = x4;
18 }
19
20
```

```
1 example1(int, int, int, int):
2     mov     xmm1, ecx
3     mov     xmm2, ecx
4     mov     xmm0, edi
5     mov     xmm1, esi
6     punpckldq    xmm1, xmm2
7     punpckldq    xmm0, xmm3
8     punpckldq    xmm0, xmm1
9     movaps  XMMWORD PTR a[rip], xmm0
10    ret
11
12 d:
13     .zero 8192
14
15 c:
16     .zero 8192
17
18 b:
19     .zero 8192
20
21 a:
22     .zero 8192
```

```
1 example1(int, int, int, int):
2     mov     dword ptr [rip + a], esi
3     mov     dword ptr [rip + a+4], esi
4     mov     dword ptr [rip + a+8], edx
5     mov     dword ptr [rip + a+12], ecx
6     ret
7
8 a:
9     .zero 8192
10
11 b:
12     .zero 8192
13
14 c:
15     .zero 8192
16
17 d:
18     .zero 8192
```

Fig. 1

Source code on the left, GCC assembly in the middle and Clang assembly on the right. Clang makes the right choice in not performing SLP. From testing the assembly on the right is 26% faster than the assembly in the middle.

```
1 #define N 1024
2 #define M 32
3 #define K 4
4
5 #define ALIGNED16 __attribute__((aligned(16)))
6
7 int a[M*2] ALIGNED16;
8 int b[M*2] ALIGNED16;
9 int c[M*2] ALIGNED16;
10 int d[M*2] ALIGNED16;
11
12 __attribute__((noinline))
13 void example1(int x1, int x2, int x3, int x4) {
14     a[0] = x1;
15     a[1] = x2;
16     a[2] = x3;
17     a[3] = x4;
18
19     b[0] = x1;
20     b[1] = x2;
21     b[2] = x3;
22     b[3] = x4;
23
24     c[0] = x1;
25     c[1] = x2;
26     c[2] = x3;
27     c[3] = x4;
28
29     d[0] = x1;
30     d[1] = x2;
31     d[2] = x3;
32     d[3] = x4;
33 }
34
```

```
1 example1(int, int, int, int):
2     mov     xmm1, ecx
3     mov     xmm2, ecx
4     mov     xmm0, edi
5     mov     xmm1, esi
6     punpckldq    xmm1, xmm2
7     punpckldq    xmm0, xmm3
8     punpckldq    xmm0, xmm1
9     movaps  XMMWORD PTR a[rip], xmm0
10    movaps  XMMWORD PTR b[rip], xmm0
11    movaps  XMMWORD PTR c[rip], xmm0
12    movaps  XMMWORD PTR d[rip], xmm0
13    ret
14
15 d:
16     .zero 8192
17
18 b:
19     .zero 8192
20
21 a:
22     .zero 8192
```

```
1 example1(int, int, int, int):
2     mov     dword ptr [rip + a], esi
3     mov     dword ptr [rip + a+4], esi
4     mov     dword ptr [rip + a+8], edx
5     mov     dword ptr [rip + a+12], ecx
6     mov     dword ptr [rip + b], esi
7     mov     dword ptr [rip + b+4], esi
8     mov     dword ptr [rip + b+8], edx
9     mov     dword ptr [rip + b+12], ecx
10    mov     dword ptr [rip + c], esi
11    mov     dword ptr [rip + c+4], esi
12    mov     dword ptr [rip + c+8], edx
13    mov     dword ptr [rip + c+12], ecx
14    mov     dword ptr [rip + d], esi
15    mov     dword ptr [rip + d+4], esi
16    mov     dword ptr [rip + d+8], edx
17    mov     dword ptr [rip + d+12], ecx
18    ret
19
20 a:
21     .zero 8192
22
23 b:
24     .zero 8192
25
26 c:
27     .zero 8192
28
29 d:
30     .zero 8192
```

Fig. 2

Source code on the left, GCC assembly in the middle and Clang assembly on the right. Clang makes the wrong choice in not performing SLP. From testing the assembly on the right is 240% slower than the assembly in the middle.

References:

1. [Vectorization Cost Modeling for NEON, AVX and SVE★](#)
2. [NeuroVectorizer: End-to-End Vectorization with Deep Reinforcement Learning](#)