IXPUG 2023

SYCL Single-Task Loop Vectorization

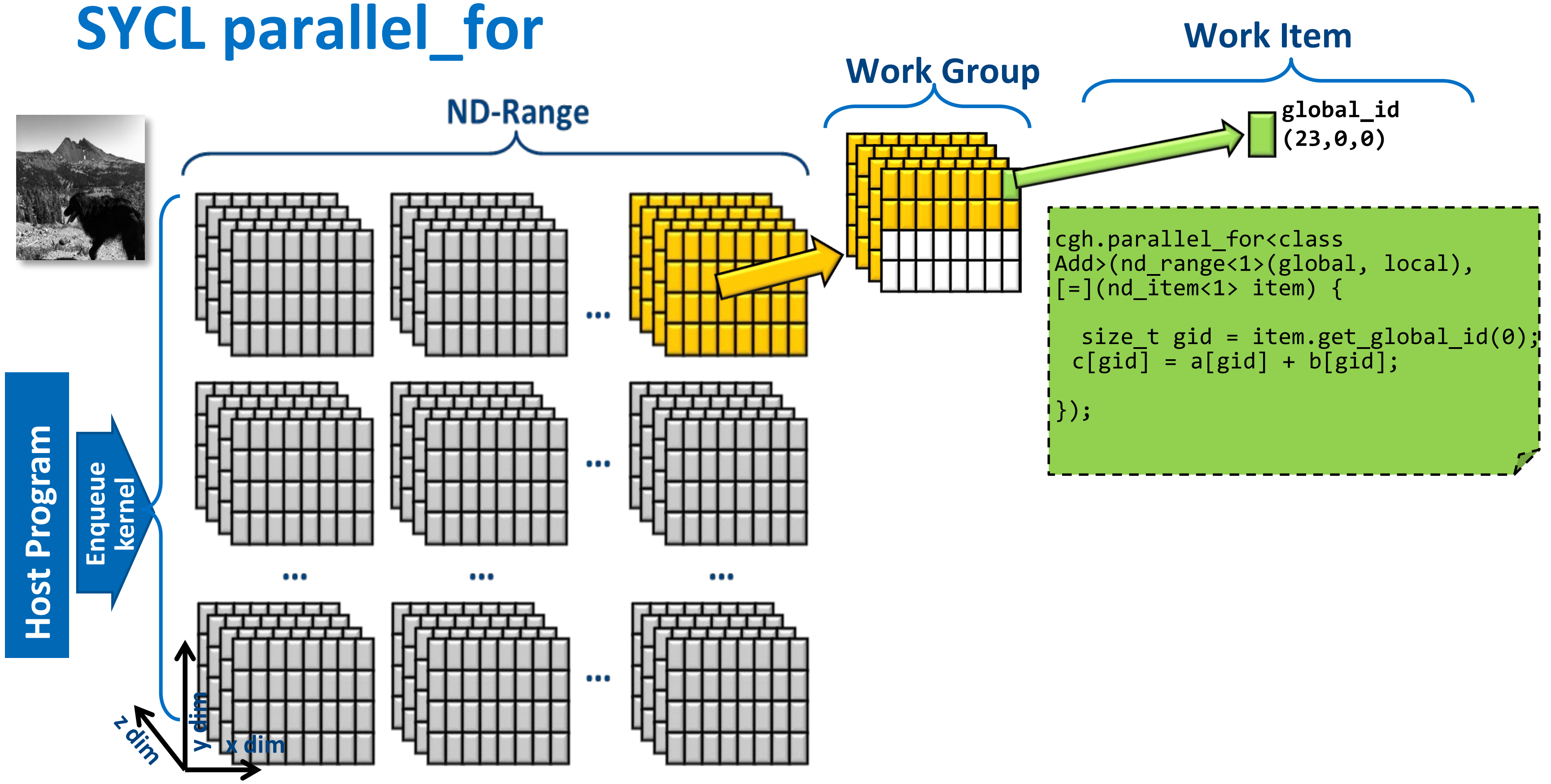
Wenju He, Wenwan Xing, Wenjuan Xu

Intel DSE Compiler Engineering

Sep 2023

Background

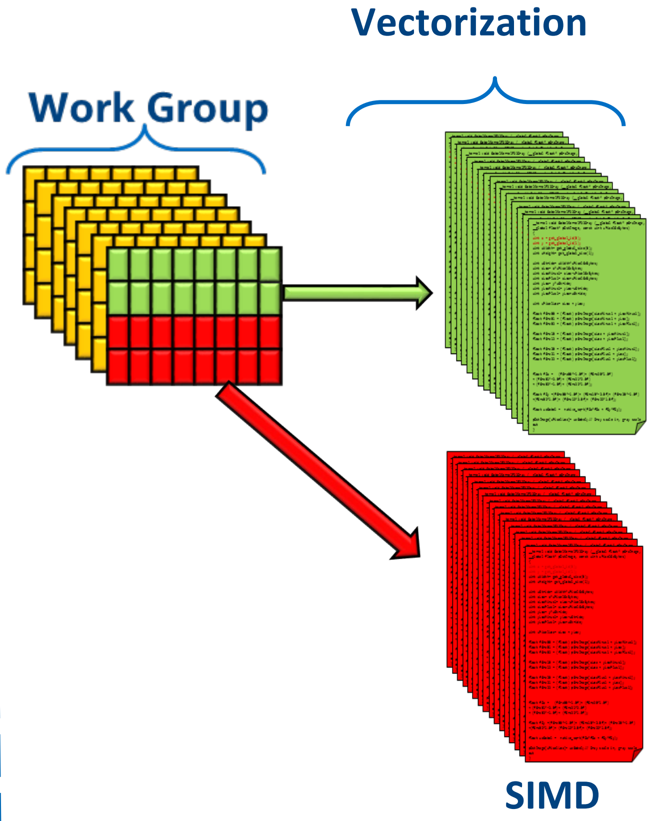
SYCL parallel_for



```
cgh.parallel_for<class
Add>(nd_range<1>(global, local),
[=](nd_item<1> item) {

  size_t gid = item.get_global_id(0);
 c[gid] = a[gid] + b[gid];

});
```

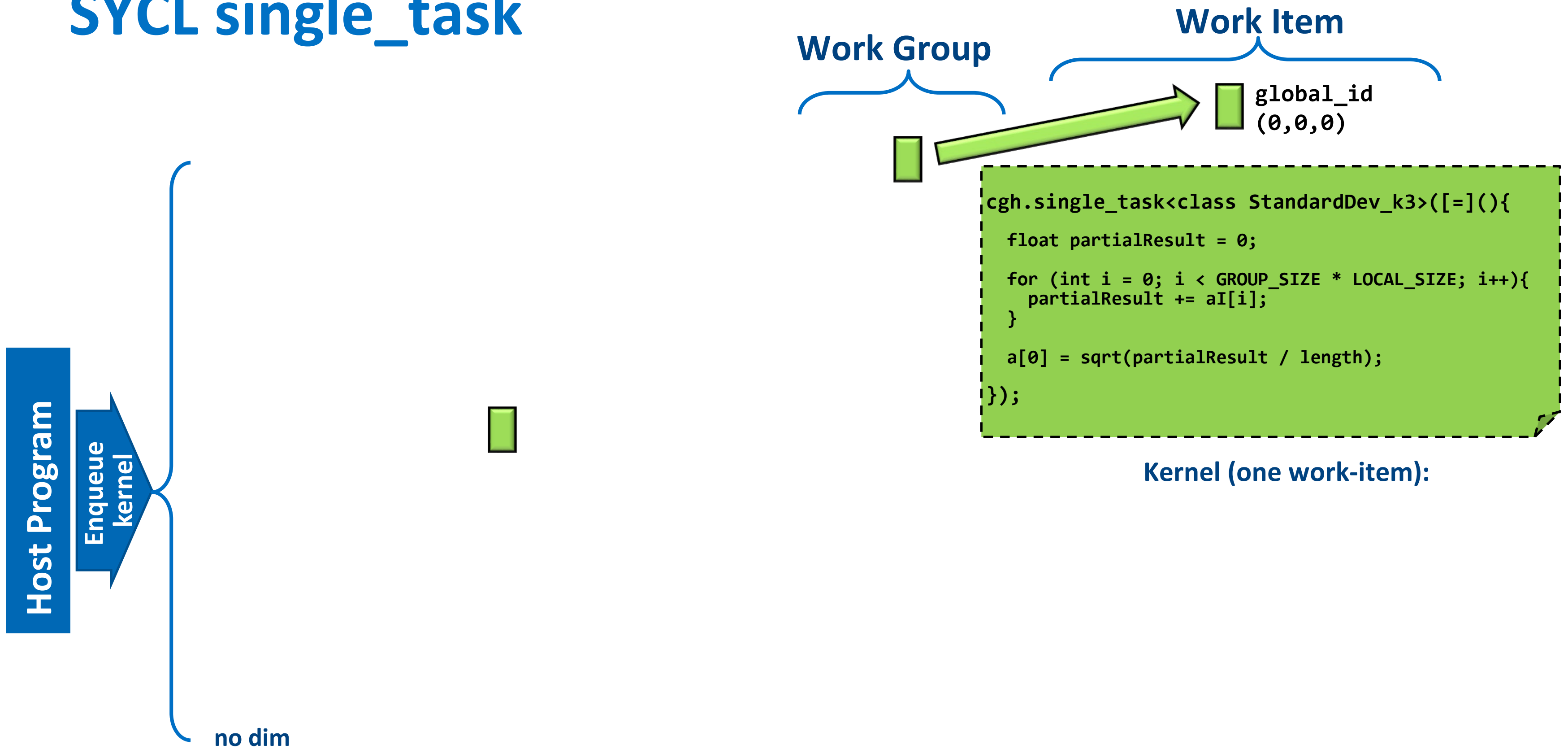


SYCL execution model is hierarchy of iteration spaces

intel.

Background

SYCL single_task



```
cgh.single_task<class StandardDev_k3>([=](){
  float partialResult = 0;

  for (int i = 0; i < GROUP_SIZE * LOCAL_SIZE; i++){
    partialResult += aI[i];
  }

  a[0] = sqrt(partialResult / length);
});
```

Kernel (one work-item):

Single workitem

- Only a single instance
- Execute once
- On single compute unit

- A single work-group
- one work-item

Loop is not light-weighted

No Vectorization

Background

- SYCL single_task
  - Usage
    - Perform a reduction of computed results of a previous SYCL parallel_for kernel.
      The result stays on device memory and could be re-used by a following-up kernel.
  - Example
    - kernel source code in SYCL dnnbench-normalization benchmark:

```
// Compute sum of tmpData and std variance
sycl::event queue_event = queue.submit([&](sycl::handler &cgh) {
    auto aI = tmpData.template get_access<sycl::access::mode::read>(cgh);
    auto aO = stdData.template get_access<sycl::access::mode::discard_write>(cgh);
    cgh.single_task<class StandardDev_k3>([=]() {
        float partialResult = 0;
        for (int i = 0; i < GROUP_SIZE * LOCAL_SIZEL; i++) {
            partialResult += aI[i];
        }
        aO[0] = sqrt(partialResult / length);
    });
});
```

GROUP_SIZE	2048
LOCAL_SIZE	16

    - If there are compute-intensive loop inside single_task, it is still worthy to vectorize it.

Background

- For SYCL parallel_for, we create a SIMD loop over kernel function. Each lane of SIMD instruction is a single work-item.

- SYCL single_task kernels won’t be vectorized with only one work-item.

- We experiment on single_task vectorization for SYCL CPU device.
  - Identify single_task kernel functions in LLVM IR
  - Insert OpenMP SIMD region directive over the loop

Identify single_task in LLVM IR

- A kernel function is deduced as single_task kernel if it doesn't contain builtins in following table

Builtins	Why
get_global_id/get_local_id/get_linear_id	There are multiple workitems
subgroup builtins workgroup builtins barrier atomic builtins	Requires synchronization
pipe builtins	Requires serialization. SIMD is forbidden

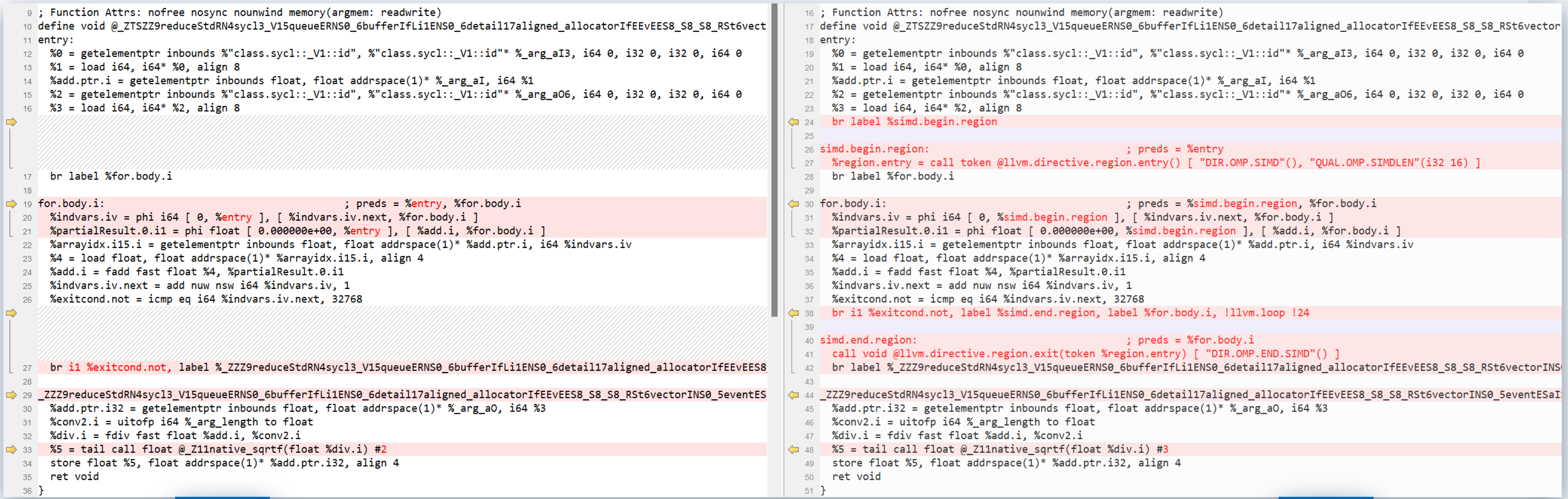
Vectorize single_task

Vectorization Steps

- Add LoopSimplifyPass into pipeline.
  - Canonicalize loops, make the following analyses simpler.
- Add a new LLVM pass to insert OMP SIMD directive to top-level loops.
  - Bail out if there are multiple loop exiting basic blocks
  - Create a new entering block to insert OpenMP SIMD region entry directive
  - Create a new exit block to insert OpenMP SIMD region exit direction.
  - Add vector-variants for SYCL builtin calls in the loops.
- Vectorization length (SIMD width) is deduced from the target Instruction Set Architecture (ISA).
- The loops will be vectorized by a loop vectorizer.
  - The vectorizer will do legality analysis and bails out in unsupported cases like data dependency inside loop.
  - If the vectorizer fails to vectorize a function, remove SIMD directives.

Insert OMP SIMD directive

- SIMD width = 16



```
; Function Attrs: nofree nosync nounwind memory(argmem: readwrite)
define void @_ZTSZZ9reduceStdRN4sycl3_V15queueERNS0_6bufferIfLi1ENS0_6detail17aligned_allocatorIfEEvEES8_S8_S8_RSt6vect
entry:
  %0 = getelementptr inbounds %"class.sycl::_V1::id", %"class.sycl::_V1::id"* %_arg_aI3, i64 0, i32 0, i32 0, i64 0
  %1 = load i64, i64* %0, align 8
  %add.ptr.i = getelementptr inbounds float, float addrspace(1)* %_arg_aI, i64 %1
  %2 = getelementptr inbounds %"class.sycl::_V1::id", %"class.sycl::_V1::id"* %_arg_aO6, i64 0, i32 0, i32 0, i64 0
  %3 = load i64, i64* %2, align 8
  br label %for.body.i

for.body.i:                                       ; preds = %entry, %for.body.i
  %indvars.iv = phi i64 [ 0, %entry ], [ %indvars.iv.next, %for.body.i ]
  %partialResult.0.i1 = phi float [ 0.000000e+00, %entry ], [ %add.i, %for.body.i ]
  %arrayidx.i15.i = getelementptr inbounds float, float addrspace(1)* %add.ptr.i, i64 %indvars.iv
  %4 = load float, float addrspace(1)* %arrayidx.i15.i, align 4
  %add.i = fadd fast float %4, %partialResult.0.i1
  %indvars.iv.next = add nuw nsw i64 %indvars.iv, 1
  %exitcond.not = icmp eq i64 %indvars.iv.next, 32768
  br i1 %exitcond.not, label %_ZZZ9reduceStdRN4sycl3_V15queueERNS0_6bufferIfLi1ENS0_6detail17aligned_allocatorIfEEvEES8

_ZZZ9reduceStdRN4sycl3_V15queueERNS0_6bufferIfLi1ENS0_6detail17aligned_allocatorIfEEvEES8_S8_S8_RSt6vectorINS0_5eventES
  %add.ptr.i32 = getelementptr inbounds float, float addrspace(1)* %_arg_aO, i64 %3
  %conv2.i = uitofp i64 %_arg_length to float
  %div.i = fdiv fast float %add.i, %conv2.i
  %5 = tail call float @_Z11native_sqrtf(float %div.i) #2
  store float %5, float addrspace(1)* %add.ptr.i32, align 4
  ret void
}
```

```
; Function Attrs: nofree nosync nounwind memory(argmem: readwrite)
define void @_ZTSZZ9reduceStdRN4sycl3_V15queueERNS0_6bufferIfLi1ENS0_6detail17aligned_allocatorIfEEvEES8_S8_S8_RSt6vector
entry:
  %0 = getelementptr inbounds %"class.sycl::_V1::id", %"class.sycl::_V1::id"* %_arg_aI3, i64 0, i32 0, i32 0, i64 0
  %1 = load i64, i64* %0, align 8
  %add.ptr.i = getelementptr inbounds float, float addrspace(1)* %_arg_aI, i64 %1
  %2 = getelementptr inbounds %"class.sycl::_V1::id", %"class.sycl::_V1::id"* %_arg_aO6, i64 0, i32 0, i32 0, i64 0
  %3 = load i64, i64* %2, align 8
  br label %simd.begin.region

simd.begin.region:                                ; preds = %entry
  %region.entry = call token @llvm.directive.region.entry() [ "DIR.OMP.SIMD"(), "QUAL.OMP.SIMDLEN"(i32 16) ]
  br label %for.body.i

for.body.i:                                       ; preds = %simd.begin.region, %for.body.i
  %indvars.iv = phi i64 [ 0, %simd.begin.region ], [ %indvars.iv.next, %for.body.i ]
  %partialResult.0.i1 = phi float [ 0.000000e+00, %simd.begin.region ], [ %add.i, %for.body.i ]
  %arrayidx.i15.i = getelementptr inbounds float, float addrspace(1)* %add.ptr.i, i64 %indvars.iv
  %4 = load float, float addrspace(1)* %arrayidx.i15.i, align 4
  %add.i = fadd fast float %4, %partialResult.0.i1
  %indvars.iv.next = add nuw nsw i64 %indvars.iv, 1
  %exitcond.not = icmp eq i64 %indvars.iv.next, 32768
  br i1 %exitcond.not, label %simd.end.region, label %for.body.i, !llvm.loop !24

simd.end.region:                                  ; preds = %for.body.i
  call void @llvm.directive.region.exit(token %region.entry) [ "DIR.OMP.END.SIMD"() ]
  br label %_ZZZ9reduceStdRN4sycl3_V15queueERNS0_6bufferIfLi1ENS0_6detail17aligned_allocatorIfEEvEES8_S8_S8_RSt6vectorINS

_ZZZ9reduceStdRN4sycl3_V15queueERNS0_6bufferIfLi1ENS0_6detail17aligned_allocatorIfEEvEES8_S8_S8_RSt6vectorINS0_5eventESaI
  %add.ptr.i32 = getelementptr inbounds float, float addrspace(1)* %_arg_aO, i64 %3
  %conv2.i = uitofp i64 %_arg_length to float
  %div.i = fdiv fast float %add.i, %conv2.i
  %5 = tail call float @_Z11native_sqrtf(float %div.i) #3
  store float %5, float addrspace(1)* %add.ptr.i32, align 4
  ret void
}
```

old

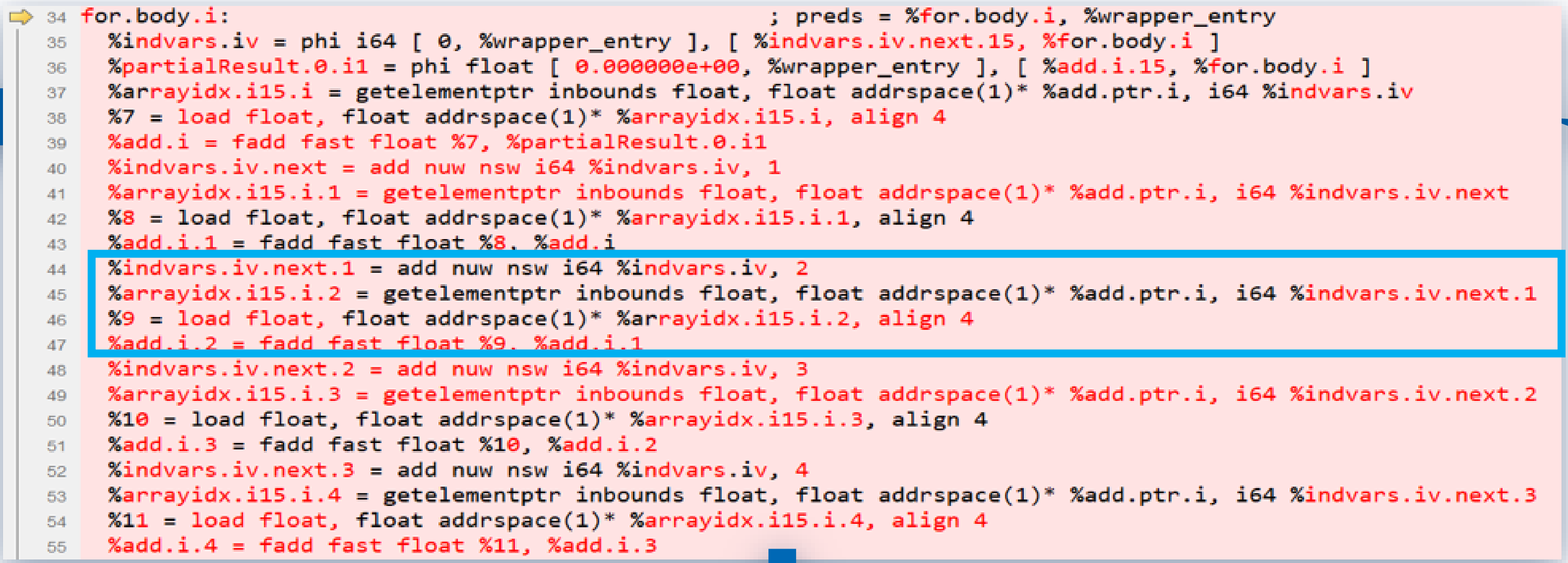
new

```
cgh.single_task<class StandardDev_k3>([=]() {
    float partialResult = 0;
    for (int i = 0; i < GROUP_SIZE * LOCAL_SIZEL; i++) {
        partialResult += aI[i];
    }
    aO[0] = sqrt(partialResult / length);
});
```

Loop IR code after Vectorizer

old

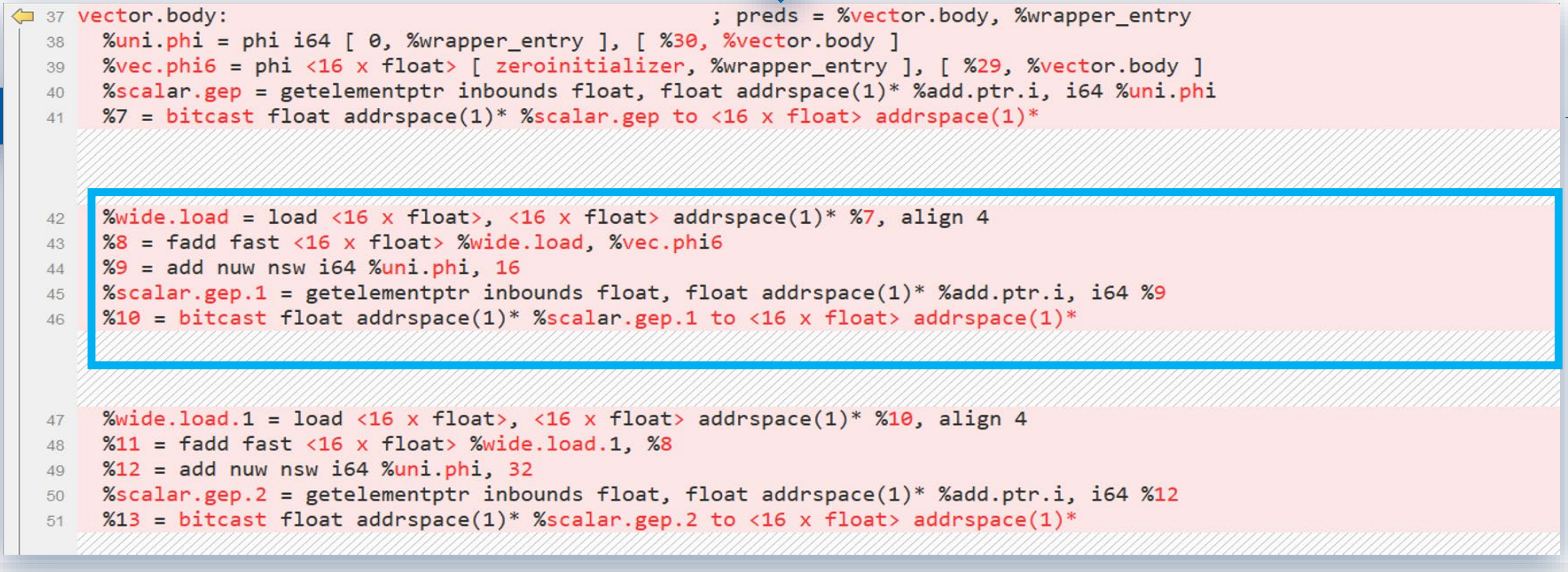
scalar code loop



```
for.body.i:                                         ; preds = %for.body.i, %wrapper_entry
  %indvars.iv = phi i64 [ 0, %wrapper_entry ], [ %indvars.iv.next.15, %for.body.i ]
  %partialResult.0.i1 = phi float [ 0.000000e+00, %wrapper_entry ], [ %add.i.15, %for.body.i ]
  %arrayidx.i15.i = getelementptr inbounds float, float addrspace(1)* %add.ptr.i, i64 %indvars.iv
  %7 = load float, float addrspace(1)* %arrayidx.i15.i, align 4
  %add.i = fadd fast float %7, %partialResult.0.i1
  %indvars.iv.next = add nuw nsw i64 %indvars.iv, 1
  %arrayidx.i15.i.1 = getelementptr inbounds float, float addrspace(1)* %add.ptr.i, i64 %indvars.iv.next
  %8 = load float, float addrspace(1)* %arrayidx.i15.i.1, align 4
  %add.i.1 = fadd fast float %8, %add.i
  %indvars.iv.next.1 = add nuw nsw i64 %indvars.iv, 2
  %arrayidx.i15.i.2 = getelementptr inbounds float, float addrspace(1)* %add.ptr.i, i64 %indvars.iv.next.1
  %9 = load float, float addrspace(1)* %arrayidx.i15.i.2, align 4
  %add.i.2 = fadd fast float %9, %add.i.1
  %indvars.iv.next.2 = add nuw nsw i64 %indvars.iv, 3
  %arrayidx.i15.i.3 = getelementptr inbounds float, float addrspace(1)* %add.ptr.i, i64 %indvars.iv.next.2
  %10 = load float, float addrspace(1)* %arrayidx.i15.i.3, align 4
  %add.i.3 = fadd fast float %10, %add.i.2
  %indvars.iv.next.3 = add nuw nsw i64 %indvars.iv, 4
  %arrayidx.i15.i.4 = getelementptr inbounds float, float addrspace(1)* %add.ptr.i, i64 %indvars.iv.next.3
  %11 = load float, float addrspace(1)* %arrayidx.i15.i.4, align 4
  %add.i.4 = fadd fast float %11, %add.i.3
```

new

vector code loop



```
vector.body:                                    ; preds = %vector.body, %wrapper_entry
  %uni.phi = phi i64 [ 0, %wrapper_entry ], [ %30, %vector.body ]
  %vec.phi6 = phi <16 x float> [ zeroinitializer, %wrapper_entry ], [ %29, %vector.body ]
  %scalar.gep = getelementptr inbounds float, float addrspace(1)* %add.ptr.i, i64 %uni.phi
  %7 = bitcast float addrspace(1)* %scalar.gep to <16 x float> addrspace(1)*

  %wide.load = load <16 x float>, <16 x float> addrspace(1)* %7, align 4
  %8 = fadd fast <16 x float> %wide.load, %vec.phi6
  %9 = add nuw nsw i64 %uni.phi, 16
  %scalar.gep.1 = getelementptr inbounds float, float addrspace(1)* %add.ptr.i, i64 %9
  %10 = bitcast float addrspace(1)* %scalar.gep.1 to <16 x float> addrspace(1)*

  %wide.load.1 = load <16 x float>, <16 x float> addrspace(1)* %10, align 4
  %11 = fadd fast <16 x float> %wide.load.1, %8
  %12 = add nuw nsw i64 %uni.phi, 32
  %scalar.gep.2 = getelementptr inbounds float, float addrspace(1)* %add.ptr.i, i64 %12
  %13 = bitcast float addrspace(1)* %scalar.gep.2 to <16 x float> addrspace(1)*
```

intel.

Builtin after vectorizer

- Another example, sin builtin is also vectorized.

```
cgh.single_task<class StandardDev_k3>([=]() {
    float partialResult = 0;
    for (int i = 0; i < chunk_size; i++) {
        partialResult += sin(aI[i]);
    }
    aO[0] = partialResult;
});
```

Scalar loop

old

```
R_X86_64_64 __ocl_svml_ze_sinf1_native
            nop word ptr cs:[rax + rax]
            vmovss  xmm0, dwordptr[r15 + 4*rsi] # xmme=mem[o],zero,zero,zero
            call  rbx
            vaddss  xmm16, xmm0, xmm16
            inc rsi
            cmp r14, rsi
            jne 0x50 <StandardDevk3+0x50>
            jmp 0x6e <StandardDev_k3+0x6e>
            vxorps  xmm16, xmm16, xmm16
            mov rax, qword ptr [rsp + 16]
            mov rcx, qword ptr [rsp + 8]
            vmovss  dword ptr[rax + 4*rcx], xmm16
```

new

Vector loop

```
R_X86_64_64 __ocl_svml_z0_sinf16_native
            nop dword ptr [rax]
            vmovups zmm0,zmmword ptr [r14+4*rbx]
            call  rsi
            vaddps  zmm16, zmm0, zmm16
            add rbx, 16
            cmp rbx, r15
            jb 0x70 <StandardDev_k3+0x70>
            vextractf64x4 ymm0, zmm16, 1
            vaddps  zmm0, zmm16, zmm0
            vextractf128  xmm1, ymm0, 1
            vaddps  xmm0, xmm0, xmm1
            vshufpd xmm1, xmm0, xmm0, 1   # xmm1 = xmme[1,0]
            vaddps  xmm0, xmm0, xmm1
            vmovshdup xmm1, xmm0     # xmm1 = xmm0[1,1,3,3]
            vaddss  xmm16, xmm0, xmm1
            mov rcx, qword ptr [rsp]
            cmp r15, rcx
            mov rsi, qword ptr [rsp + 8]
            je  0x1d5 <StandardDev_k3+0x1d5>
            mov rax, rcx
            sub rax, r15
            cmp rax, 5
            ja  0xff <StandardDev_k3+0xff>
            movabs  rbx, 0
R_X86_64_64 __ocl_svml_z0_sinf1_native
            nop word ptr [rax + rax]
            vmovss xmm0, dword ptr [r14 + 4*r15] # xmm0 = mem[0],zero,zero,zero
            vzeroupper
            call  rbx
            vaddss  xmm16, xmm0, xmm16
            inc r15
            cmp qword ptr [rsp], r15
            jne 0xe0 <StandardDev_k3+0xe0>
            jmp ox1d5 <StandardDev_k3+0x1d5>
            mov rbx, rsi
            movabs  rcx, 0
```

Scalar loop

Performance result

- Performance (SIMD width = 16)

  SYCL dnnbench-normalization

  Time before vectorization : 0.186s

  Time after vectorization : 0.146s

  Gain : 21.2%

  Device : ICX 8358

- It could be adapted to other CPU backend and GPU vector backend as well.

intel®