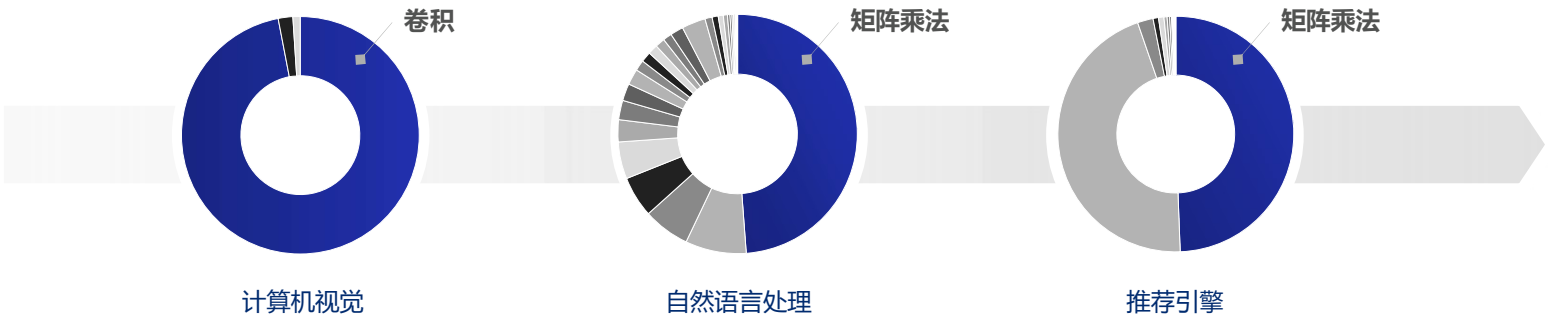
中国科技大学研究生学术报告

oneDNN Graph 编程接口和编译器

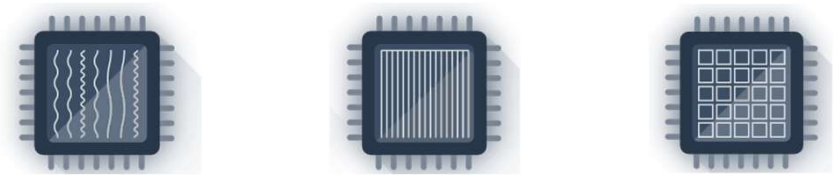
李剑慧，英特尔高级首席AI工程师

人工智能优化的驱动力

多样化的人工智能应用



快速发展的人工智能硬件



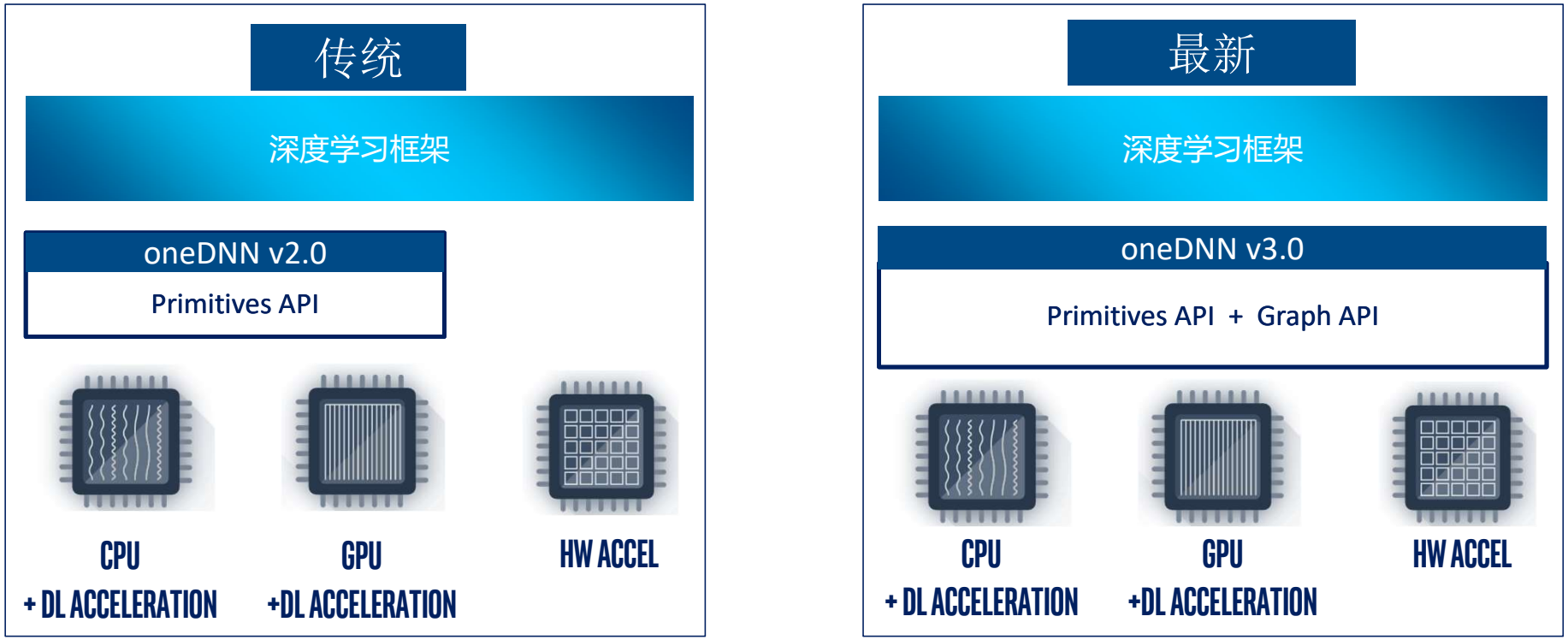
CPU+ 深度学习硬件加速

GPU
+深度学习硬件加速

深度学习专用加速芯片

仅加速卷积和矩阵乘还不能够，需要针对计算图优化

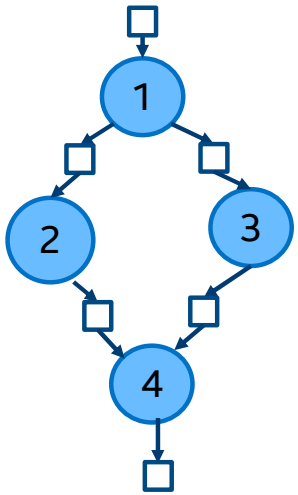
oneDNN的最新发展



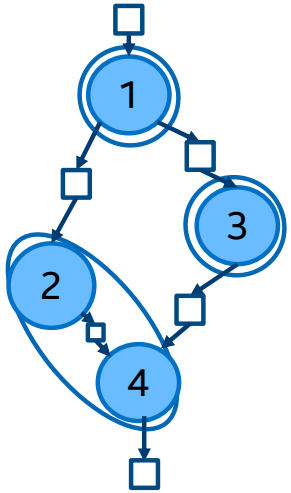
- Graph API 允许硬件后端最大限度地提高性能
- 多AI 硬件共享相同编程接口降低集成成本

https://spec.oneapi.com/onednn

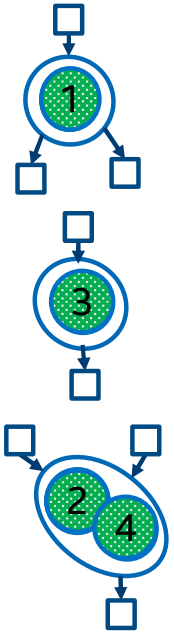
oneDNN Graph 图编程概览



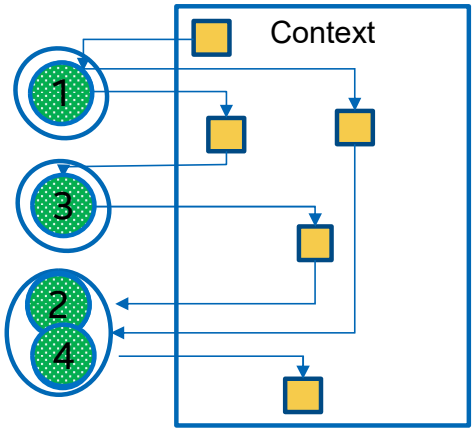
构造



分区



编译



执行

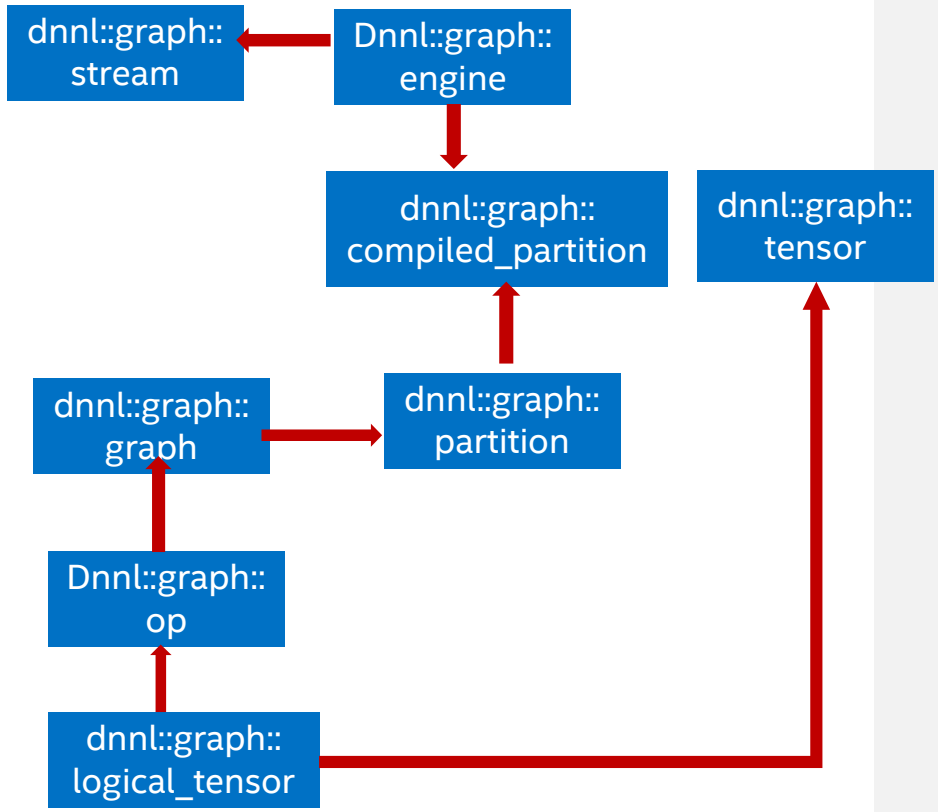
oneDNN Graph 编程模型

构造和分区

- Logical tensor: 逻辑张量, 张量的元数据，如 dims、数据类型、布局
- Op: DNN 操作，具有属性的, 与输入/输出相关联的逻辑张量
- Graph: DNN 计算图， DNN 操作和逻辑张量的集合
- Partition: 计算图分区，为特定硬件目标分割出的待优化子图

编译与执行

- Engine – 执行硬件
- Stream – 执行上下文
- Compiled partition: 计算图分区的已编译对象
- Tensor: 张量元数据 + 数据存储



图构造和分区编程接口

class logical_tensor

```
// Constructs a logical tensor object.
// dims is tensor dimensions, -1 means a particular axis of dims is unknown
logical_tensor(size_t tid, const dims_t &adims, data_type in_type,
               layout_type layout_type = layout_type::strided, layout_id layout_id = layout_id::any);
// Query the memory size of tensor.
size_t get_mem_size();
```

class op

```
// Constructs an Op object based on input/output logical tensors and kind is the computation of op like conv, batchnorm, etc.
op(size_t id, kind akind,
   const std::vector<logical_tensor> &inputs, const std::vector<logical_tensor> &outputs, const std::string &debug_string);
// Sets the attribute according to the name and type.
op &set_attr(const std::string &name, const Type &a);
```

class graph

```
// Constructs a graph for a specific device target.
graph(engine::kind engine_kind);
// Add an op to the graph to construct DAG for analysis.
bool add_op(const op &op);
// Get list of partitions from a graph.
partition_vec get_partitions(llga_partition_policy policy);
```

class partition

```
// Compile the partition based on the input/output logical tensors and the associated device info (engine).
compiled_partition compile(const std::vector<logical_tensor> &inputs, std::vector<logical_tensor> &outputs, const engine &e);
```

图构造和分区示例代码

```
    // Create logical tensor for conv
    logical_tensor conv0_src_desc {0, input_dims, data_type::f32};
    logical_tensor conv0_weight_desc {1, weight_dims, data_type::f32};
    logical_tensor conv0_bias_desc {2, bias_dims, data_type::f32};
    logical_tensor conv0_dst_desc {3, dst_dims, data_type::f32};
    logical_tensor conv0_bias_add_dst_desc {4, dst_dims,
data_type::f32};
    // Create logical tensor for bn
    logical_tensor bn0_scale_desc {5, bias_dims, data_type::f32};
    logical_tensor bn0_shift_desc {6, bias_dims, data_type::f32};
    logical_tensor bn0_mean_desc {7, bias_dims, data_type::f32};
    logical_tensor bn0_var_desc {8, bias_dims, data_type::f32};
    logical_tensor bn0_dst_desc {9, dst_dims, data_type::f32};

    // Create logical tensor for relu
    logical_tensor relu0_dst_desc {10, dst_dims, data_type::f32};

    // Create conv OP and set attributes
    op conv0(0, llga_op_kind::kConvolution,
       {conv0_src_desc, conv0_weight_desc}, {conv0_dst_desc},"conv0");
    conv0.set_attr<std::vector<int64_t>>("strides", {4, 4});
    conv0.set_attr<std::string>("data_format", "NCX");    ……
    conv0.set_attr<int64_t>("groups", 1);
    // Create biasadd OP and set attributes
    op bias_add(1, llga_op_kind::kBiasAdd,
       {conv0_dst_desc, conv0_bias_desc},
       {conv0_bias_add_dst_desc},"bias_add");
```

```
// Create bn OP and set attributes
op bn0(2, llga_op_kind::kBatchNormInference,
          {conv0_bias_add_dst_desc, bn0_scale_desc,
           bn0_shift_desc, bn0_mean_desc, bn0_var_desc},
          {bn0_dst_desc},"bn0");
bn0.set_attr<float>("epsilon", 0.f);

// Create relu OP
op relu0(3, llga_op_kind::kReLU,
          {bn0_dst_desc},
          {relu0_dst_desc},"relu0");

// Add OPs to graph
graph g {engine::kind::gpu};
g.add_op(conv0);
g.add_op(bias_add);
g.add_op(bn0);
g.add_op(relu0);
g.finalize();

// Get partitions
// fused into one parition: `conv0+biasadd+bn0+relu0`
auto partitions =
   g.get_partitions(llga_partition_policy_fusion);
```

图编译和执行接口

class compiled_partition

```
// Execute a compiled partition.
void execute(stream &astream, const std::vector<tensor> &inputs, const std::vector<tensor> &outputs);
// Query logical tensor according to the given id.
logical_tensor query_logical_tensor(uint64_t tid)
```

class tensor

```
// Construct a tensor based on given logical tensor and data
// handle
tensor(const logical_tensor &lt, void *handle);
```

class stream

```
// Constructs a stream for the specified engine.
stream(engine &aengine, const stream_attr *attr = nullptr);
// Constructs a stream for the specified engine and SYCL queue.
stream(engine &aengine, const sycl::queue &queue, const stream_attr
*attr = nullptr);
```

class engine

```
// Constructs an engine with specified kind and device id
engine(kind akind, int device_id);
// Constructs an engine from SYCL device and context.
engine(kind akind, const sycl::device &dev, const sycl::context &ctx);
// Set allocator to engine.
set_allocator(allocator &alloc);
```

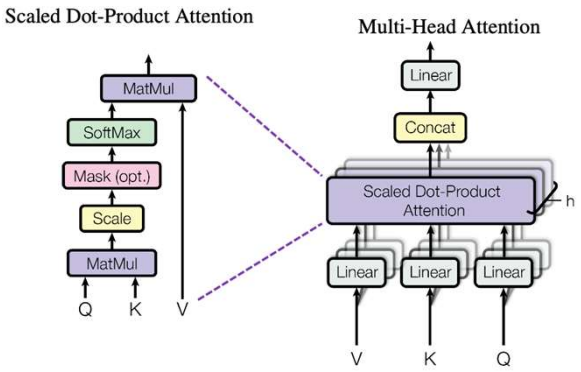
图编译和执行示例代码

```
// initialize a queue object
sycl::queue q {ctx, device0};
// constructs concrete engine object
engine eng {engine::kind::gpu, q.get_device(),q.get_context()};

// Create logical tensors with shape info
logical_tensor conv0_src_desc {0, input_dims, data_type::f32};
logical_tensor conv0_weight_desc {1, weight_dims, data_type::f32};
   ...
logical_tensor bn0_dst_desc {9, dst_dims, data_type::f32};

// compile process
compiled_partition c_partition = partition.compile(…);

// query memory size (e.g. output tensor)
auto output = c_partition.query_logical_tensor(tid);
size_t mem_size = output.get_mem_size();

// constructs concrete stream object
stream strm {eng, q};

// prepare input/output tensor
vector<tensor> inputs {{logical_tensor0, handle0}};
vector<tensor> outputs {{logical_tensor1, handle1}};
// execute process
c_partition.execute(strm, inputs, outputs);
```

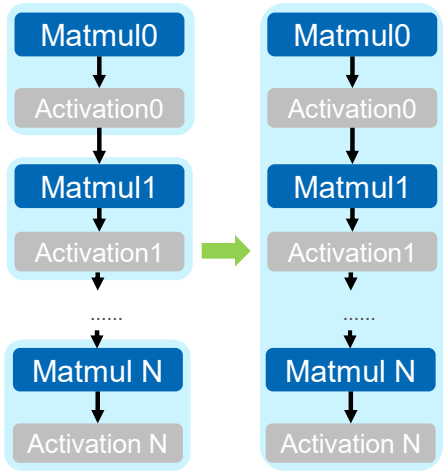
使用oneDNN加速深度学习软件

oneDNN API

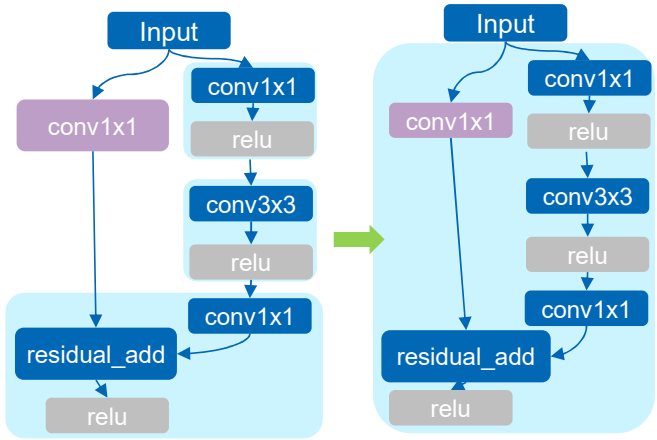
优化常用计算子图



MHA

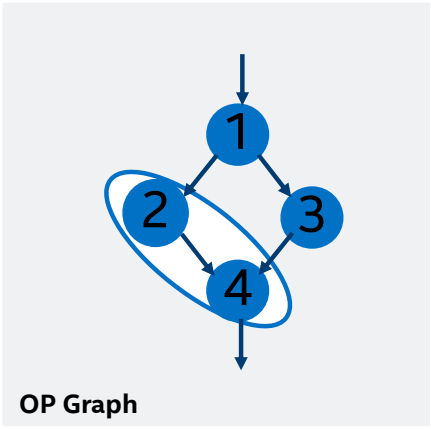


MLP



Conv Block

oneDNN Graph 编译器



OP Graph

```
Fused_asymentric_dynamic_quantization_kernel (A[m, k, mt,  kt],
B[n, k, kt, nt ], C[m, n, mt, nt])
{
   Parallel loop m_o = 0, M/MB {
         Parallel loop n_o = 0, N/NB {
              A'[m_o, 0:MB] = 0;
              Loop k_o = 0, K/KB {
                   A'[m_o, 0:MB] += A[m_o, k_o, 0:MB, 0:KB] *
   a_scale[k_o, 0:KB]
                   A'[m_o, 0:MB] *= b_zp
                   Call matmul_micro(A[m_o, k_o, 0:MB, 0:KB],
                              B[n_o, k_0, 0:NB, 0:KB],
                              C[m_o, n_o, 0:MB, 0:NB]);
              }
              C[m_o, n_o, 0:MB, 0:NB]) += A'[m_o, 0:MB]
              C[m_o, n_o, 0:MB, 0:NB]) *= b_scale/b_zp
         }
    }
)
```

Tensor IR

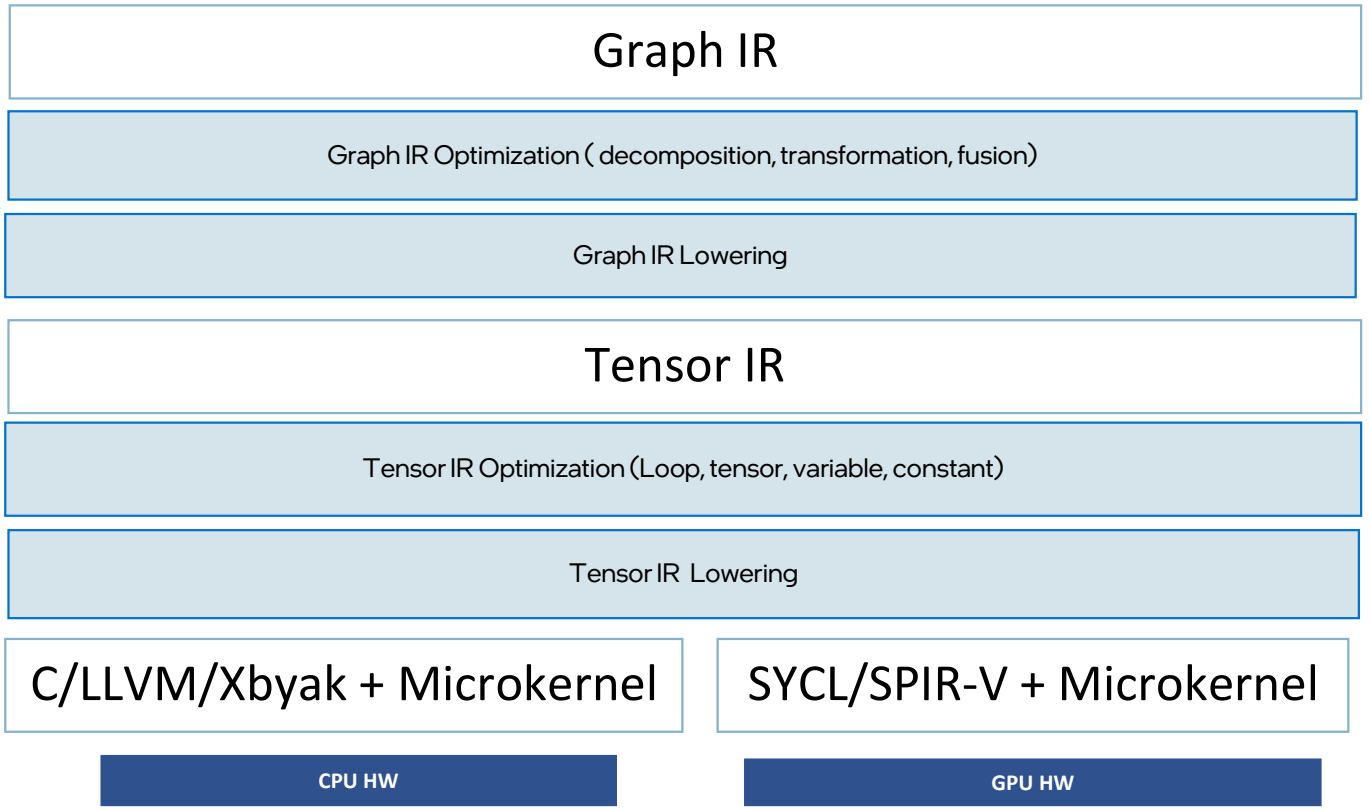
```
vmovaps  3211264(%rcx,%rax,4), %ymm1
vmovaps  3211296(%rcx,%rax,4), %ymm2
vmovaps  3211328(%rcx,%rax,4), %ymm3
vaddps  6422528(%rcx,%rax,4), %ymm1, %ymm1
vaddps  6422560(%rcx,%rax,4), %ymm2, %ymm2
vmovaps  3211360(%rcx,%rax,4), %ymm4
vaddps  6422592(%rcx,%rax,4), %ymm3, %ymm3
vaddps  6422624(%rcx,%rax,4), %ymm4, %ymm4
vmaxps  %ymm0, %ymm1, %ymm1
vmaxps  %ymm0, %ymm2, %ymm2
vmaxps  %ymm0, %ymm3, %ymm3
vmovaps  %ymm1, 6422528(%rcx,%rax,4)
vmovaps  %ymm2, 6422560(%rcx,%rax,4)
vmaxps  %ymm0, %ymm4, %ymm1
vmovaps  %ymm3, 6422592(%rcx,%rax,4)
vmovaps  %ymm1, 6422624(%rcx,%rax,4)
addq  $32, %rax
```

Binary Code

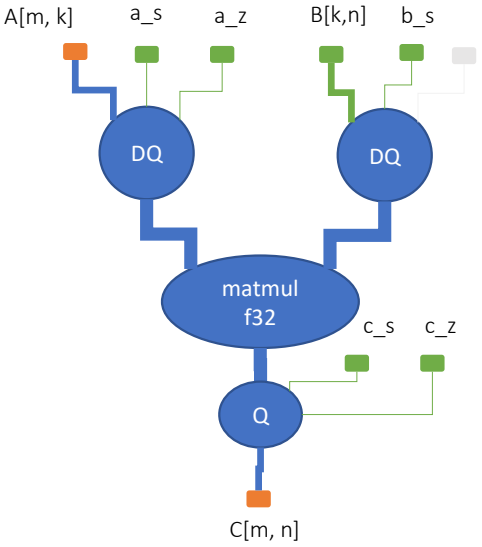
- 底级深度学习编译器，优化生成代码效率
- 结合编译器技术和手动调优内核的经验
- 自动生成算子代码，并进一步为子图生成代码

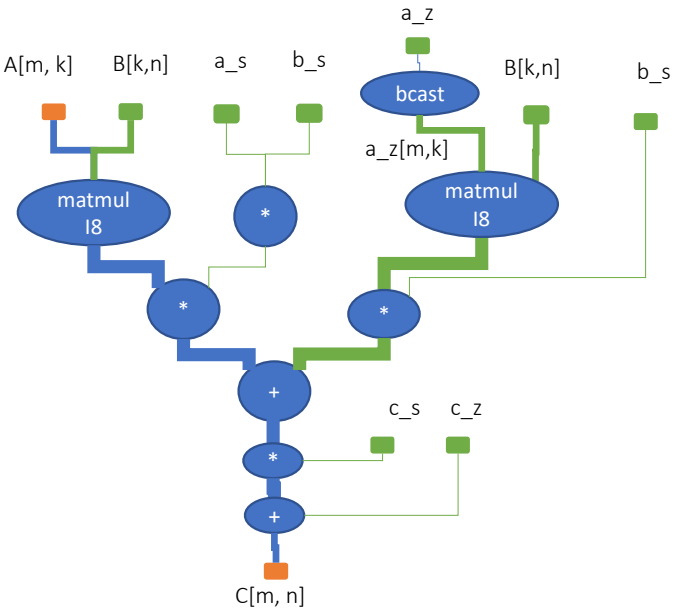
https://arxiv.org/abs/2301.01333

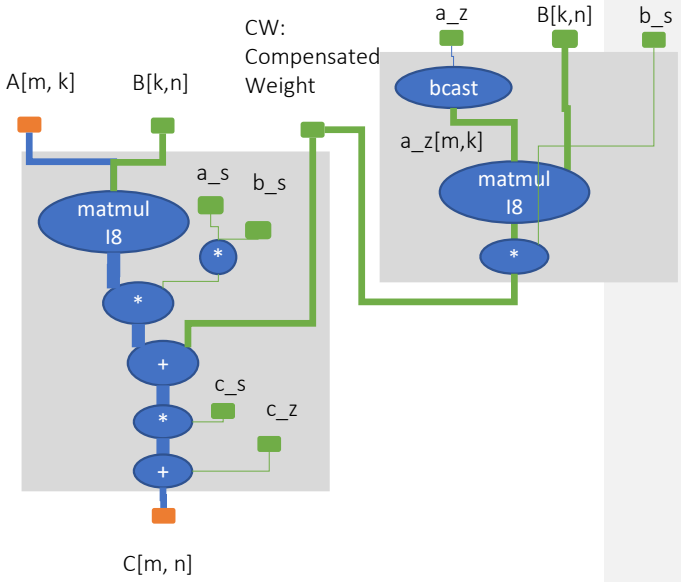
oneDNN Graph 编译器内部表示



图优化



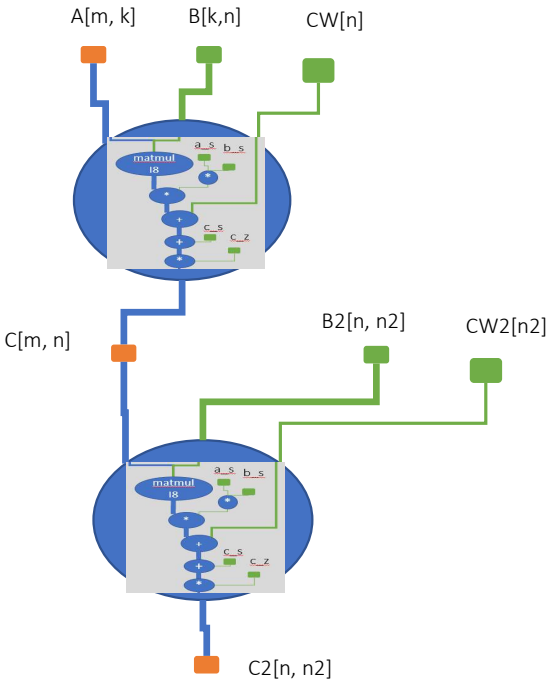




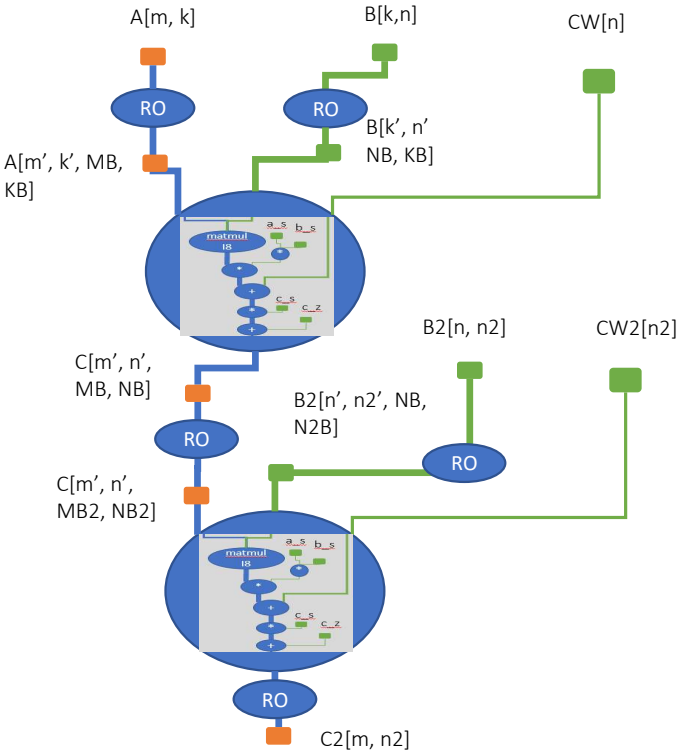
Low-Precision Conversion

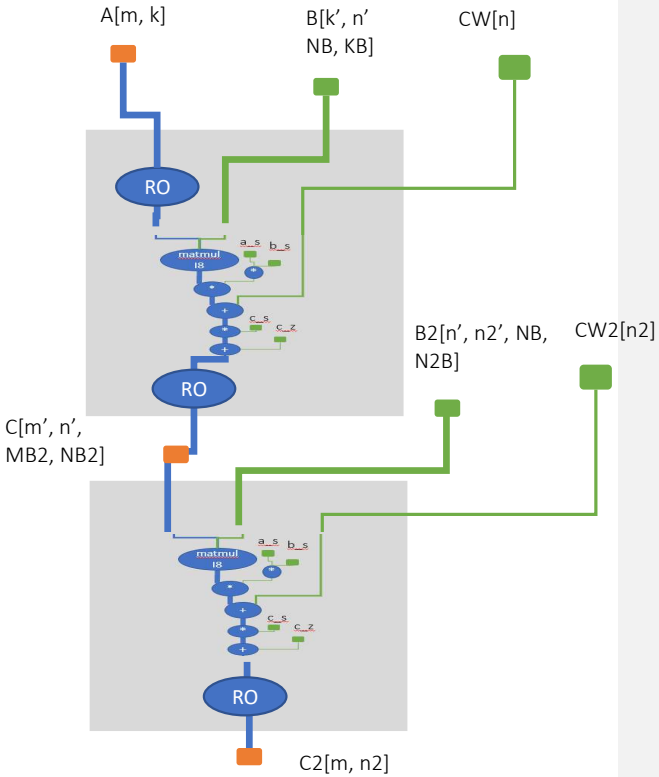
Const Weight Preprocess

图优化（续）



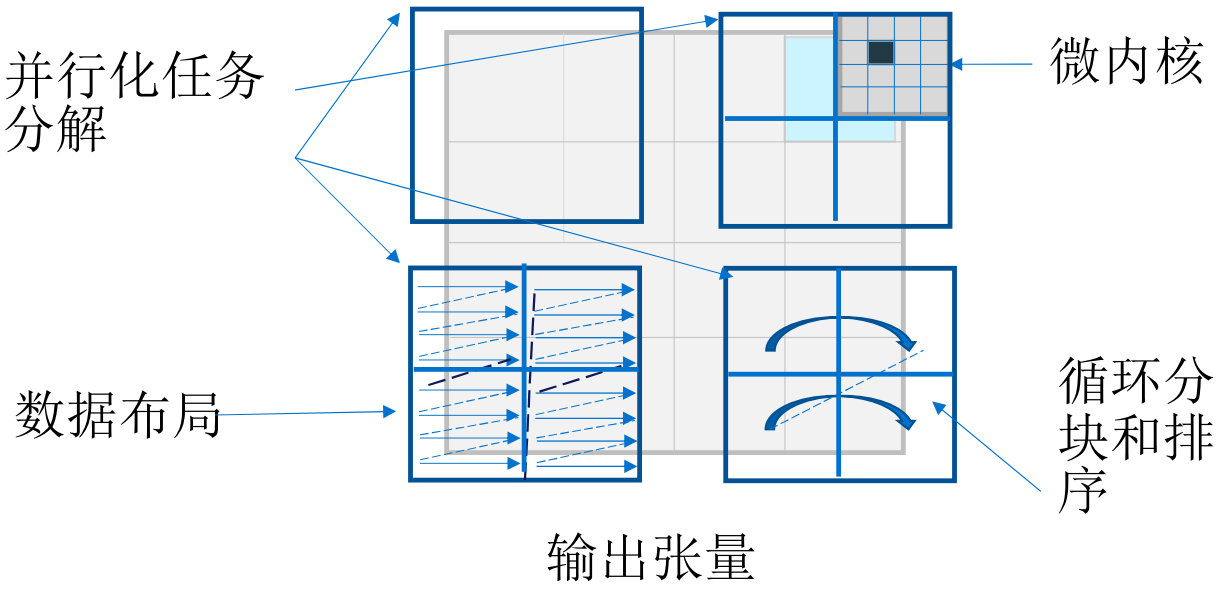
Layout Propagation





Fusion

高效内核生成的要点

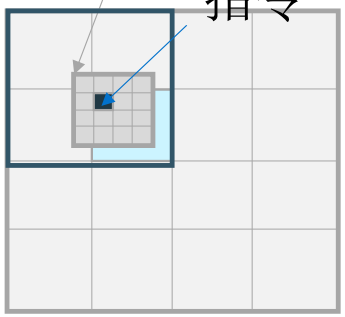


可调超参数	受影响硬件效率
单核任务大小	多核系统
微内核大小	矢量/矩阵硬件单元
循环分块尺寸 循环顺序	缓存/TLB/内存
数据布局分块尺寸	缓存/TLB/内存

微内核

微内核

矢量/矩阵指令



$$C[M,N] = \sum_{i=0}^{batch} A_i[M,K] * B_i[K,N]$$

Batch-Reduce gemm interface

High-Performance Deep Learning via a Single Building Block, Evangelos Georganas, Alexander Heinecke

微内核 代码示例

$$C[32,32] = \sum_{i=0}^{16} A_i[32,32] * B_i[32,32]$$

```
Batch_reduce_gemm(A[0:32, 0:32],
                  B[0:32, 0:32],
                  C[0:32, 0:32],
                  batch = 16)
{
  tileloaddt1(tmm0, C);
  tileloaddt1(tmm1, C + 64);
  tileloaddt1(tmm2, C + 2048);
  tileloaddt1(tmm3, C + 2112);
  for (int i = 0; i < batch; i++) {
    tileloaddt1(tmm4, A);
    tileloaddt1(tmm5, A + 1024);
    tileloaddt1(tmm6, B);
    tdbpf16ps(tmm0, tmm4, tmm6);
    tdbpf16ps(tmm2, tmm5, tmm6);
    tileloaddt1(tmm7, B + 64);
    tdbpf16ps(tmm1, tmm4, tmm7);
    tdbpf16ps(tmm3, tmm5, tmm7);
    A += 1024; B += 1024;
  }
  tilestored(C, tmm0);
  tilestored(C + 64, tmm1);
  tilestored(C + 2048, tmm2);
  tilestored(C + 2112, tmm3);
}
```

专门用于特定的张量形状

与硬件矢量/矩阵大小对齐

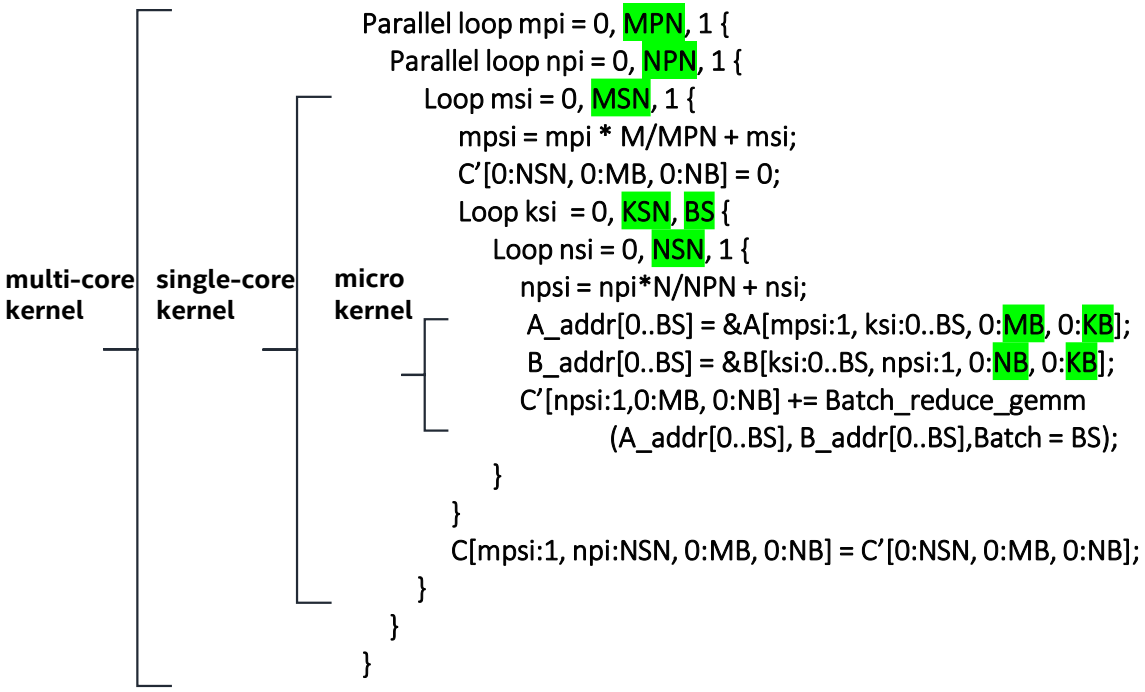
循环展开、软件流水线、预取

矢量化/张量化

寄存器分块和分配

针对特点硬件手工精心遍制的代码模板

矩阵乘算子模板



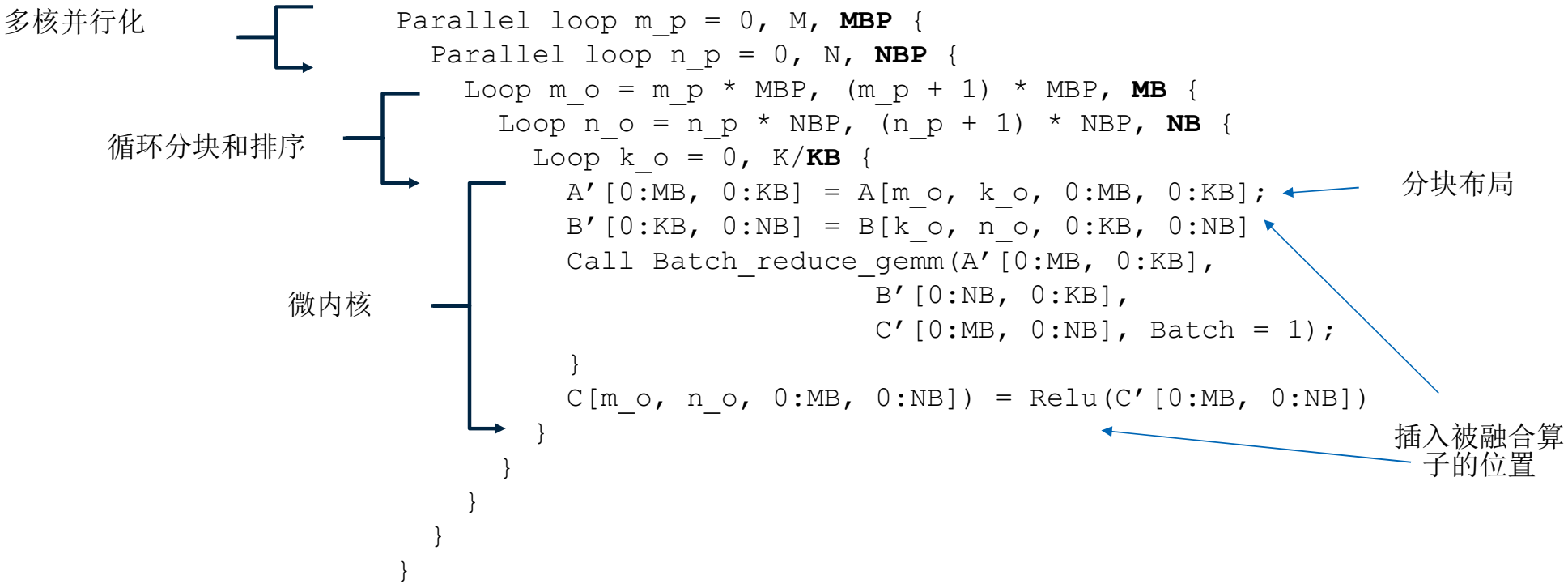
- 模板描述了卷积和矩阵乘的算法
- 启发式算法决定模板的参数
- 最内层循环中使用微内核
  - 简化AI编译器设计，专注于外层循环
  - 提供硬件无关的接口，来抽象不同的AI加速指令

融合算子模板

```
Parallel loop mpi = 0, MPN, 1 {
  Parallel loop npi = 0, NPN, 1 {
    Loop msi = 0, MSN, 1 {
      mpsi = mpi * M/MPN + msi;
      C'[0:NSN, 0:MB, 0:NB] = 0;
      Loop ksi  = 0, KSN, BS {
        Loop nsi = 0, NSN, 1 {
          npsi = npi*N/NPN + nsi;
          A_addr[0..BS] = &A[mpsi:1, ksi:0..BS, 0:MB, 0:KB];
          B_addr[0..BS] = &B[ksi:0..BS, npsi:1, 0:NB, 0:KB];
          C'[npsi:1,0:MB, 0:NB] += Batch_reduce_gemm
                (A_addr[0..BS], B_addr[0..BS],Batch = BS);
        }
      }
      C[mpsi:1, npi:NSN, 0:MB, 0:NB] = C'[0:NSN, 0:MB, 0:NB];
    }
  }
}
```

```
Parallel loop mpi = 0, MPN, 1 {
  pre_op_anchor#1 : A[mpi*MSN:MSN, 0:KSN, 0:MB, 0:KB];
  pre_op_anchor#1 : B[0:KSN, 0:NPSN, 0:NB, 0:KB];
  Parallel loop npi = 0, NPN, 1 {
    pre_op_anchor#2 : A[mpi*MSN:MSN, 0:KSN, 0:MB, 0:KB];
    pre_op_anchor#2 : B[0:KSN,npi*NSN:NSN, 0:NB, 0:KB];
    Loop msi = 0, MSN, 1 {
      mpsi = mpi * M/MPN + msi;
      pre_op_anchor#3 : A[mpsi:1, 0:KSN, 0:MB, 0:KB];
      pre_op_anchor#3 : B[0:KSN,npi*NSN:NSN, 0:NB, 0:KB];
      C'[0:NSN, 0:MB, 0:NB] = 0;
      Loop ksi = 0, KSN, BS {
        pre_op_anchor#4 : A[mpsi:1, ksi:BS, 0:MB, 0:KB];
        pre_op_anchor#4 : B[ksi:BS, npi*NSN:NSN, 0:NB, 0:KB];
        Loop nsi = 0, NSN, 1 {
          npsi = npi*N/NPN + nsi;
          pre_op_anchor#5 : A[mpsi:1, ksi:BS, 0:MB, 0:KB];
          pre_op_anchor#5 : B[ksi:BS, npsi:1, 0:NB, 0:KB];
          A_addr[0..BS] = &A[mpsi:1, ksi: 0..BS, 0:MB, 0:KB];
          B_addr[0..BS] = &B[ksi:0..BS, npsi:1, 0:NB, 0:KB];
          C'[nsi:1,0:MB, 0:NB] += Batch_reduce_gemm
                (A_addr[0..BS], B_addr[0..BS],Batch = BS);
        }
      }
      C[mpsi:1, npi:NSN, 0:MB, 0:NB] = C'[0:NSN, 0:MB, 0:NB];
      post_op_anchor#1 : C[mpsi:1, npi:NSN, 0:MB, 0:NB];
    }
    post_op_anchor#2 : C[mpi*MSN:MSN, npi*NSN:NSN, 0:MB, 0:NB];
  }
  post_op_anchor#3 : C[mpi*MSN:MSN, 0:NPSN, 0:MB, 0:NB];
}
```

融合算子的生成代码



使用良好启发式方法来将问题分解为单核内核和微内核、循环顺序和数据布局

总结

- AI**软硬件快速发展，需要针对计算图优化**
- oneDNN Graph**统一**AI**硬件上底层性能加速库的编程接口，支持与**AI**框架和**SYCL**编程互操作**
- oneDNN Graph**编译器结合编译技术和手动调优内核, 针对常见AI计算子图生成高效代码**

Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. Results may vary.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details.
No product or component can be absolutely secure.

Slide 50 - **Texas Advanced Computing Center (TACC) Frontera references**
Article: *HPCWire: Visualization & Filesystem Use Cases Show Value of Large Memory Fat Notes on Frontera*.
www.intel.com/content/dam/support/us/en/documents/memory-and-storage/data-center-persistent-mem/Intel-Optane-DC-Persistent-Memory-Quick-Start-Guide.pdf
software.intel.com/content/www/us/en/develop/articles/introduction-to-programming-with-persistent-memory-from-intel.html
wreda.github.io/papers/assise-osdi20.pdf
KFBIO
KFBIO m. tuberculosis screening detectron2 model throughput performance on 2nd Intel® Xeon® Gold 6252 processor: NEW: Test 1 (single instance with PyTorch 1.6: Tested by Intel as of 5/22/2020. 2-socket 2nd Gen Intel® Xeon® Gold 6252 Processor, 24 cores, HT On, Turbo ON, Total Memory 192 GB (12 slots/16 GB/2666 MHz), BIOS: SSE5C620.86B.02.01.0008.031920191559 (ucode: 0x500002c), Ubuntu 18.04.4 LTS, kernel 5.3.0-51-generic, mitigated Test 2 (24 instances with PyTorch 1.6: Tested by Intel as of 5/22/2020. 2-socket 2nd Gen Intel Xeon Gold 6252 Processor, 24 cores, HT On, Turbo ON, Total Memory 192 GB (12 slots/16 GB/2666 MHz), BIOS: SSE5C620.86B.02.01.0008.031920191559 (ucode: 0x500002c), Ubuntu 18.04.4 LTS, kernel 5.3.0-51-generic, mitigated BASELINE: (single instance with PyTorch 1.4): Tested by Intel as of 5/22/2020. 2-socket 2nd Gen Intel Xeon Gold 6252 Processor, 24 cores, HT On, Turbo ON, Total Memory 192 GB (12 slots/16 GB/2666 MHz), BIOS: SSE5C620.86B.02.01.0008.031920191559 (ucode: 0x500002c), Ubuntu 18.04.4 LTS, kernel 5.3.0-51-generic, mitigated.

Tangent Studios
Configurations for Render Times with Intel® Embree, testing conducted by Tangent Animation Labs. Render farm: 8x Intel® Core™ processors +hyperthread*2 + 128gig. In-office workstations: Intel® Xeon® processors HP blade c7000 chassis, with HP460 gen8 blades - 2x Intel Xeon E5-2650 V2, Eight Core 2.6GHz-128GB. Software: Blender 2.78 with custom build using Intel® Embree. For more information on Tangent's work with Embree, watch this video: www.youtube.com/watch?time_continue=251&v=_2Ia4h8q3xs&feature=emb_logo
Recreation of the performance numbers can be recreated using Agent327, Blender and Embree.

Chaos Group - Up to 90% Memory Reduction for Displacement
Testing conducted by Chaos Group with Intel® Embree 2020. Software Corona Renderer 5 with Intel Embree. Up to 90% memory reduction calculated using Corona Renderer 5 with regular displacement grids per triangle of 154 bytes versus Corona Renderer 5 with Intel Embree, which has a displacement capability grid of 12 bytes per grid triangle. (12/154 = 7.8% usage or >90% memory reduction.) Recreation of the performance numbers can be accomplished using Corona Renderer 5 and Embree. For more information, visit the Corona Renderer Blog: blog.corona-renderer.com/corona-renderer-5-for-3ds-max-released/

The Addams Family 2 - Gained a 10% to 20%—and sometimes 25%—efficiency in rendering, saving thousands of hours in rendering production time.
Testing Date: Results are based on data conducted by Cinesite 2020-21. 10% to up to 25% rendering efficiency/thousands of hours saved in rendering production time/15 hrs per frame per shot to 12-13 hrs.
Cinesite Configuration: 18-core Intel® Xeon® Scalable processors (W-2295) used in render farm, 2nd gen Intel Xeon processor-based workstations (W-2135 and -2195) used. Rendering tools: Gaffer, Arnold, along with optimizations by Intel® Open Image Denoise.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.