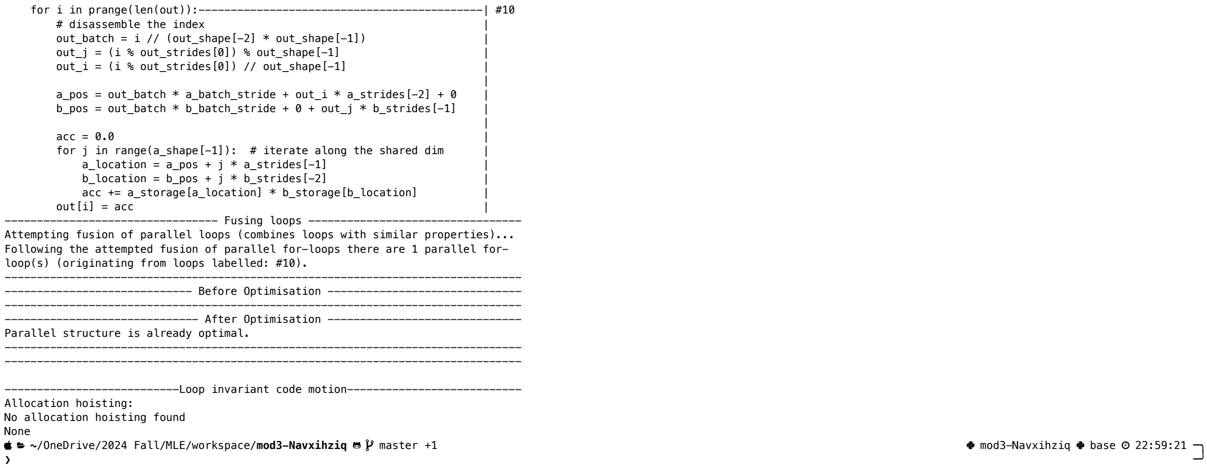
```
Last login: Tue Nov 19 22:42:06 on ttys009
❯ cd ~/OneDrive/2024\ Fall/MLE/workspace/mod3-Navxihziq
❯ clear
❯ source .venv/bin/activate
❯ python project/parallel_check.py
MAP
OMP: Info #276: omp_set_nested routine deprecated, please use omp_set_max_active_levels instead.

================================================================================
 Parallel Accelerator Optimizing:  Function tensor_map.<locals>._map,
/Users/qizhixuan/Library/CloudStorage/OneDrive-Personal/2024
Fall/MLE/workspace/mod3-Navxihziq/minitorch/fast_ops.py (163)
================================================================================


Parallel loop listing for  Function tensor_map.<locals>._map, /Users/qizhixuan/Library/CloudStorage/OneDrive-Personal/2024 Fall/MLE/workspace/mod3-Navxihziq/minitorch/fast_ops.py (163)
--------------------------------------------------------------------------------|loop #ID
    def _map(                                                                   |
        out: Storage,                                                           |
        out_shape: Shape,                                                       |
        out_strides: Strides,                                                   |
        in_storage: Storage,                                                    |
        in_shape: Shape,                                                        |
        in_strides: Strides,                                                    |
    ) -> None:                                                                  |
        # TODO: Implement for Task 3.1.                                         |
        # check if out, in are stride-aligned                                   |
        # if out_strides == in_strides:                                         |
        #     for i in prange(len(out)):                                        |
        #         out[i] = fn(in_storage[i])                                    |
        # else:                                                                 |
        # TODO: check if out, in are stride-aligned                             |
        # coerce the shape to int32                                             |
        out_shape = out_shape.astype(np.int32)                                  |
        in_shape = in_shape.astype(np.int32)                                    |
        for i in prange(len(out)):----------------------------------------------| #2
            out_index = np.zeros(len(out_shape), dtype=np.int32)  # buffer------| #0
            in_index = np.zeros(len(in_shape), dtype=np.int32)  # buffer--------| #1
            to_index(i, out_shape, out_index)                                   |
            broadcast_index(out_index, out_shape, in_shape, in_index)           |
            out[i] = fn(in_storage[index_to_position(in_index, in_strides)])    |
--------------------------------- Fusing loops ---------------------------------
Attempting fusion of parallel loops (combines loops with similar properties)...
Following the attempted fusion of parallel for-loops there are 3 parallel for-
loop(s) (originating from loops labelled: #2, #0, #1).
--------------------------------------------------------------------------------
---------------------------- Optimising loop nests -----------------------------
Attempting loop nest rewrites (optimising for the largest parallel loops)...

+--2 is a parallel loop
   +--0 --> rewritten as a serial loop
   +--1 --> rewritten as a serial loop
--------------------------------------------------------------------------------
----------------------------- Before Optimisation ------------------------------
Parallel region 0:
+--2 (parallel)
   +--0 (parallel)
   +--1 (parallel)


--------------------------------------------------------------------------------
------------------------------ After Optimisation ------------------------------
Parallel region 0:
+--2 (parallel)
   +--0 (serial)
   +--1 (serial)



Parallel region 0 (loop #2) had 0 loop(s) fused and 2 loop(s) serialized as part
 of the larger parallel loop (#2).
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

---------------------------Loop invariant code motion---------------------------
Allocation hoisting:
The memory allocation derived from the instruction at
/Users/qizhixuan/Library/CloudStorage/OneDrive-Personal/2024
Fall/MLE/workspace/mod3-Navxihziq/minitorch/fast_ops.py (182) is hoisted out of
the parallel loop labelled #2 (it will be performed before the loop is executed
and reused inside the loop):
   Allocation:: out_index = np.zeros(len(out_shape), dtype=np.int32)  # buffer
    - numpy.empty() is used for the allocation.
The memory allocation derived from the instruction at
/Users/qizhixuan/Library/CloudStorage/OneDrive-Personal/2024
Fall/MLE/workspace/mod3-Navxihziq/minitorch/fast_ops.py (183) is hoisted out of
the parallel loop labelled #2 (it will be performed before the loop is executed
and reused inside the loop):
   Allocation:: in_index = np.zeros(len(in_shape), dtype=np.int32)  # buffer
    - numpy.empty() is used for the allocation.
None
ZIP

================================================================================
 Parallel Accelerator Optimizing:  Function tensor_zip.<locals>._zip,
/Users/qizhixuan/Library/CloudStorage/OneDrive-Personal/2024
Fall/MLE/workspace/mod3-Navxihziq/minitorch/fast_ops.py (214)
================================================================================


Parallel loop listing for  Function tensor_zip.<locals>._zip, /Users/qizhixuan/Library/CloudStorage/OneDrive-Personal/2024 Fall/MLE/workspace/mod3-Navxihziq/minitorch/fast_ops.py (214)
------------------------------------------------------------------------------|loop #ID
    def _zip(                                                                 |
        out: Storage,                                                         |
        out_shape: Shape,                                                     |
        out_strides: Strides,                                                 |
        a_storage: Storage,                                                   |
        a_shape: Shape,                                                       |
        a_strides: Strides,                                                   |
        b_storage: Storage,                                                   |
        b_shape: Shape,                                                       |
        b_strides: Strides,                                                   |
    ) -> None:                                                                |
        # TODO: Implement for Task 3.1.                                       |
        # coerce the shape to int32                                           |
        out_shape = out_shape.astype(np.int32)                                |
        a_shape = a_shape.astype(np.int32)                                    |
        b_shape = b_shape.astype(np.int32)                                    |
        # TODO: check if out, a, b are stride-aligned                         |
        # if (                                                                |
        #     len(out_shape) == len(a_shape) == len(b_shape)                  |
        #     and np.array_equal(out_shape, a_shape)                          |
        #     and np.array_equal(out_shape, b_shape)                          |
        #     and np.array_equal(out_strides, a_strides)                      |
        #     and np.array_equal(out_strides, b_strides)                      |
        # ):                                                                  |
        #     for i in prange(len(out)):                                      |
        #         out[i] = fn(a_storage[i], b_storage[i])                     |
        # else:                                                               |
        for i in prange(len(out)):--------------------------------------------| #6
            out_index = np.zeros(len(out_shape), dtype=np.int32)  # buffer----| #3
            a_index = np.zeros(len(a_shape), dtype=np.int32)  # buffer--------| #4
            b_index = np.zeros(len(b_shape), dtype=np.int32)  # buffer--------| #5
            to_index(i, out_shape, out_index)                                 |
            broadcast_index(out_index, out_shape, a_shape, a_index)           |
            broadcast_index(out_index, out_shape, b_shape, b_index)           |
                                                                              |
            out[i] = fn(                                                      |
                a_storage[index_to_position(a_index, a_strides)],             |
                b_storage[index_to_position(b_index, b_strides)],             |
            )                                                                 |
--------------------------------- Fusing loops ---------------------------------
Attempting fusion of parallel loops (combines loops with similar properties)...
Following the attempted fusion of parallel for-loops there are 4 parallel for-
loop(s) (originating from loops labelled: #6, #3, #4, #5).
--------------------------------------------------------------------------------
---------------------------- Optimising loop nests -----------------------------
Attempting loop nest rewrites (optimising for the largest parallel loops)...

+--6 is a parallel loop
   +--3 --> rewritten as a serial loop
   +--4 --> rewritten as a serial loop
   +--5 --> rewritten as a serial loop
--------------------------------------------------------------------------------
----------------------------- Before Optimisation ------------------------------
Parallel region 0:
+--6 (parallel)
   +--3 (parallel)
   +--4 (parallel)
   +--5 (parallel)


--------------------------------------------------------------------------------
------------------------------ After Optimisation ------------------------------
Parallel region 0:
+--6 (parallel)
   +--3 (serial)
   +--4 (serial)
   +--5 (serial)



Parallel region 0 (loop #6) had 0 loop(s) fused and 3 loop(s) serialized as part
 of the larger parallel loop (#6).
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

---------------------------Loop invariant code motion---------------------------
Allocation hoisting:
The memory allocation derived from the instruction at
/Users/qizhixuan/Library/CloudStorage/OneDrive-Personal/2024
Fall/MLE/workspace/mod3-Navxihziq/minitorch/fast_ops.py (242) is hoisted out of
the parallel loop labelled #6 (it will be performed before the loop is executed
and reused inside the loop):
   Allocation:: out_index = np.zeros(len(out_shape), dtype=np.int32)  # buffer
    - numpy.empty() is used for the allocation.
The memory allocation derived from the instruction at
/Users/qizhixuan/Library/CloudStorage/OneDrive-Personal/2024
Fall/MLE/workspace/mod3-Navxihziq/minitorch/fast_ops.py (243) is hoisted out of
the parallel loop labelled #6 (it will be performed before the loop is executed
and reused inside the loop):
   Allocation:: a_index = np.zeros(len(a_shape), dtype=np.int32)  # buffer
    - numpy.empty() is used for the allocation.
The memory allocation derived from the instruction at
/Users/qizhixuan/Library/CloudStorage/OneDrive-Personal/2024
Fall/MLE/workspace/mod3-Navxihziq/minitorch/fast_ops.py (244) is hoisted out of
the parallel loop labelled #6 (it will be performed before the loop is executed
and reused inside the loop):
   Allocation:: b_index = np.zeros(len(b_shape), dtype=np.int32)  # buffer
    - numpy.empty() is used for the allocation.
None
REDUCE

================================================================================
 Parallel Accelerator Optimizing:  Function tensor_reduce.<locals>._reduce,
/Users/qizhixuan/Library/CloudStorage/OneDrive-Personal/2024
Fall/MLE/workspace/mod3-Navxihziq/minitorch/fast_ops.py (279)
================================================================================


Parallel loop listing for  Function tensor_reduce.<locals>._reduce, /Users/qizhixuan/Library/CloudStorage/OneDrive-Personal/2024 Fall/MLE/workspace/mod3-Navxihziq/minitorch/fast_ops.py (279)
----------------------------------------------------------------------------------------|loop #ID
    def _reduce(                                                                        |
        out: Storage,                                                                   |
        out_shape: Shape,                                                               |
        out_strides: Strides,                                                           |
        a_storage: Storage,                                                             |
        a_shape: Shape,                                                                 |
        a_strides: Strides,                                                             |
        reduce_dim: int,                                                                |
    ) -> None:                                                                          |
        # TODO: Implement for Task 3.1.                                                 |
        # coerce the shape to int32                                                     |
        out_shape = out_shape.astype(np.int32)                                          |
        a_shape = a_shape.astype(np.int32)                                              |
        for i in prange(len(out)):------------------------------------------------------| #9
            out_index = np.zeros(len(out_shape), dtype=np.int32)  # buffer--------------| #7
            a_index = np.zeros(len(a_shape), dtype=np.int32)  # buffer------------------| #8
            to_index(i, out_shape, out_index)                                           |
            # copy the out_index to the a_index (except for the reduce dim)             |
            for j in range(len(a_shape) - 1):                                           |
                j = j if j < reduce_dim else j + 1                                      |
```

~~~
                a_index[j] = out_index[j]                                               |
                                                                                        |
            a_index[reduce_dim] = 0                                                     |
            a_pos = index_to_position(a_index, a_strides)                               |
            temp: float = a_storage[a_pos]  # avoid inner access to global variable     |
            for j in range(1, a_shape[reduce_dim]):                                     |
                temp = fn(temp, float(a_storage[a_pos + j * a_strides[reduce_dim]]))    |
            out[i] = temp                                                               |
--------------------------------- Fusing loops ---------------------------------
Attempting fusion of parallel loops (combines loops with similar properties)...
Following the attempted fusion of parallel for-loops there are 3 parallel for-
loop(s) (originating from loops labelled: #9, #7, #8).
--------------------------------------------------------------------------------
---------------------------- Optimising loop nests -----------------------------
Attempting loop nest rewrites (optimising for the largest parallel loops)...

+--9 is a parallel loop
   +--8 --> rewritten as a serial loop
   +--7 --> rewritten as a serial loop
--------------------------------------------------------------------------------
----------------------------- Before Optimisation ------------------------------
Parallel region 0:
+--9 (parallel)
   +--8 (parallel)
   +--7 (parallel)


--------------------------------------------------------------------------------
------------------------------ After Optimisation ------------------------------
Parallel region 0:
+--9 (parallel)
   +--8 (serial)
   +--7 (serial)



Parallel region 0 (loop #9) had 0 loop(s) fused and 2 loop(s) serialized as part
 of the larger parallel loop (#9).
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

---------------------------Loop invariant code motion---------------------------
Allocation hoisting:
The memory allocation derived from the instruction at
/Users/qizhixuan/Library/CloudStorage/OneDrive-Personal/2024
Fall/MLE/workspace/mod3-Navxihziq/minitorch/fast_ops.py (293) is hoisted out of
the parallel loop labelled #9 (it will be performed before the loop is executed
and reused inside the loop):
   Allocation:: out_index = np.zeros(len(out_shape), dtype=np.int32)  # buffer
    - numpy.empty() is used for the allocation.
The memory allocation derived from the instruction at
/Users/qizhixuan/Library/CloudStorage/OneDrive-Personal/2024
Fall/MLE/workspace/mod3-Navxihziq/minitorch/fast_ops.py (294) is hoisted out of
the parallel loop labelled #9 (it will be performed before the loop is executed
and reused inside the loop):
   Allocation:: a_index = np.zeros(len(a_shape), dtype=np.int32)  # buffer
    - numpy.empty() is used for the allocation.
None
MATRIX MULTIPLY

================================================================================
 Parallel Accelerator Optimizing:  Function _tensor_matrix_multiply,
/Users/qizhixuan/Library/CloudStorage/OneDrive-Personal/2024
Fall/MLE/workspace/mod3-Navxihziq/minitorch/fast_ops.py (311)
================================================================================


Parallel loop listing for  Function _tensor_matrix_multiply, /Users/qizhixuan/Library/CloudStorage/OneDrive-Personal/2024 Fall/MLE/workspace/mod3-Navxihziq/minitorch/fast_ops.py (311)
--------------------------------------------------------------------------|loop #ID
def _tensor_matrix_multiply(                                              |
    out: Storage,                                                         |
    out_shape: Shape,                                                     |
    out_strides: Strides,                                                 |
    a_storage: Storage,                                                   |
    a_shape: Shape,                                                       |
    a_strides: Strides,                                                   |
    b_storage: Storage,                                                   |
    b_shape: Shape,                                                       |
    b_strides: Strides,                                                   |
) -> None:                                                                |
    """NUMBA tensor matrix multiply function.                             |
                                                                          |
    Should work for any tensor shapes that broadcast as long as           |
                                                                          |
    ```                                                                   |
    assert a_shape[-1] == b_shape[-2]                                     |
    ```                                                                   |
                                                                          |
    Optimizations:                                                        |
                                                                          |
    * Outer loop in parallel                                              |
    * No index buffers or function calls                                  |
    * Inner loop should have no global writes, 1 multiply.                |
                                                                          |
                                                                          |
    Args:                                                                 |
    ----                                                                  |
        out (Storage): storage for `out` tensor                           |
        out_shape (Shape): shape for `out` tensor                         |
        out_strides (Strides): strides for `out` tensor                   |
        a_storage (Storage): storage for `a` tensor                       |
        a_shape (Shape): shape for `a` tensor                             |
        a_strides (Strides): strides for `a` tensor                       |
        b_storage (Storage): storage for `b` tensor                       |
        b_shape (Shape): shape for `b` tensor                             |
        b_strides (Strides): strides for `b` tensor                       |
                                                                          |
    Returns:                                                              |
    -------                                                               |
        None : Fills in `out`                                             |
                                                                          |
    """                                                                   |
    a_batch_stride = a_strides[0] if a_shape[0] > 1 else 0                |
    b_batch_stride = b_strides[0] if b_shape[0] > 1 else 0                |
                                                                          |
    # TODO: Implement for Task 3.2.                                       |
~~~



```
    for i in prange(len(out)):--------------------------------------------| #10
        # disassemble the index                                           |
        out_batch = i // (out_shape[-2] * out_shape[-1])                  |
        out_j = (i % out_strides[0]) % out_shape[-1]                      |
        out_i = (i % out_strides[0]) // out_shape[-1]                     |
                                                                          |
        a_pos = out_batch * a_batch_stride + out_i * a_strides[-2] + 0    |
        b_pos = out_batch * b_batch_stride + 0 + out_j * b_strides[-1]    |
                                                                          |
        acc = 0.0                                                         |
        for j in range(a_shape[-1]):  # iterate along the shared dim      |
            a_location = a_pos + j * a_strides[-1]                        |
            b_location = b_pos + j * b_strides[-2]                        |
            acc += a_storage[a_location] * b_storage[b_location]          |
        out[i] = acc                                                      |
--------------------------------- Fusing loops ---------------------------------
Attempting fusion of parallel loops (combines loops with similar properties)...
Following the attempted fusion of parallel for-loops there are 1 parallel for-
loop(s) (originating from loops labelled: #10).
--------------------------------------------------------------------------------
----------------------------- Before Optimisation ------------------------------
--------------------------------------------------------------------------------
------------------------------ After Optimisation ------------------------------
Parallel structure is already optimal.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

---------------------------Loop invariant code motion----------------------------
Allocation hoisting:
No allocation hoisting found
None
 ~/OneDrive/2024 Fall/MLE/workspace/mod3-Navxihziq   master +1                                                           mod3-Navxihziq  base ⊙ 22:59:21 ─╮
❯
```