# A Appendix

## A.1 Code

The code for benchmarks and experiments is available at `https://github.com/awav/gambit`, and the fork of TensorFlow repository with extension to XLA compiler (eXLA) is available at `https://github.com/awav/tensorflow`.

## A.2 Additional experiments

| Dataset | Distance | n | d | KeOps | eJAX | eTF | JAX | TF |
|---|---|---|---|---|---|---|---|---|
| Random | $L^2$ | 10000 | 100 | 983263 | 277364 | 284777 | 281695 | 280826 |
| Random | $L^2$ | 10000 | 10 | 2587001 | 291751 | 295029 | 287958 | 294168 |
| Random | $L^2$ | 10000 | 3 | 3662188 | 292804 | 294971 | 288098 | 294776 |
| Random | $L^2$ | 1000000 | 100 | 24367 | 2433 | 2530 | $\varnothing$ | $\varnothing$ |
| Random | $L^2$ | 1000000 | 10 | 106726 | 2505 | 2601 | $\varnothing$ | $\varnothing$ |
| Random | $L^2$ | 1000000 | 3 | 123765 | 2512 | 2605 | $\varnothing$ | $\varnothing$ |
| Random | $L^2$ | 10000000 | 100 | 2461 | 243 | 253 | $\varnothing$ | $\varnothing$ |
| Random | $L^2$ | 10000000 | 10 | 11546 | 251 | 261 | $\varnothing$ | $\varnothing$ |
| Random | $L^2$ | 10000000 | 3 | 13192 | 251 | 261 | $\varnothing$ | $\varnothing$ |
| Random | $L^1$ | 1000000 | 100 | 24307 | 517 | 521 | $\varnothing$ | $\varnothing$ |
| Random | $L^1$ | 1000000 | 10 | 108739 | 2494 | 2590 | $\varnothing$ | $\varnothing$ |
| Random | Cosine | 1000000 | 100 | 32520 | 2434 | 2515 | $\varnothing$ | $\varnothing$ |
| Random | Cosine | 1000000 | 10 | 106876 | 2507 | 2612 | $\varnothing$ | $\varnothing$ |
| MNIST | $L^2$ | 60000 | 784 | 41084 | 32290 | 33455 | 25544 | 26138 |
| MNIST | $L^1$ | 60000 | 784 | 40697 | 2356 | 2985 | 2498 | 2988 |
| Fashion | $L^2$ | 60000 | 784 | 40399 | 32382 | 33428 | 25558 | 26128 |
| Fashion | $L^1$ | 60000 | 784 | 40982 | 2357 | 2984 | 2498 | 2989 |
| Glove-50 | Cosine | 1183514 | 50 | 3464257 | 2103 | 1929 | $\varnothing$ | $\varnothing$ |
| Glove-100 | Cosine | 1183514 | 100 | 631420 | 2053 | 1871 | $\varnothing$ | $\varnothing$ |
| Glove-200 | Cosine | 1183514 | 200 | 398293 | 1967 | 1724 | $\varnothing$ | $\varnothing$ |

Table 3: Query processing rates (queries per second) for kNN. $n$ and $d$ are the number of data points and the data dimension respectively. Runs which failed due to memory overflow are denoted by $\varnothing$. Runs with eXLA are denoted eJAX and eTF respectively.

Figures 5 and 6 depict XLA HLO graphs for kernel matrix-vector multiplication before and after splitting optimisation in eXLA optimisation pipeline (section 4.3), respectively. The same configuration of the kernel is used as in section 5.1, i.e. squared exponential kernel from Matthews et al. (2017). By kernel matrix-vector multiplication expression we mean the function $g(\mathbf{x}, \mathbf{y}, \mathbf{v}) = k(\mathbf{x}, \mathbf{y})\mathbf{v}$, where $k(\mathbf{x}, \mathbf{y}) = \sigma^2 \exp\left(-1/2\|\mathbf{x} - \mathbf{y}\|^2 / l^2\right)$ is the kernel with $\sigma^2$ and $l$ hyperparameters. The size of 1-dimensional input vectors $\mathbf{v}$, $\mathbf{x}$ and $\mathbf{y}$ is $1\mathrm{e}{-6}$. In turn, the size of the corresponding kernel matrix is $1\mathrm{e}{-6} \times 1\mathrm{e}{-6}$, and in double precision would require to allocate 8TB. The *tensor size threshold* was set to 1GB, and eXLA splitting optimisation pass divided the expression of the kernel matrix-vector multiplication into smaller chunks, such that the maximum tensor size in the graph is $125 \times 1\mathrm{e}{-6}$.

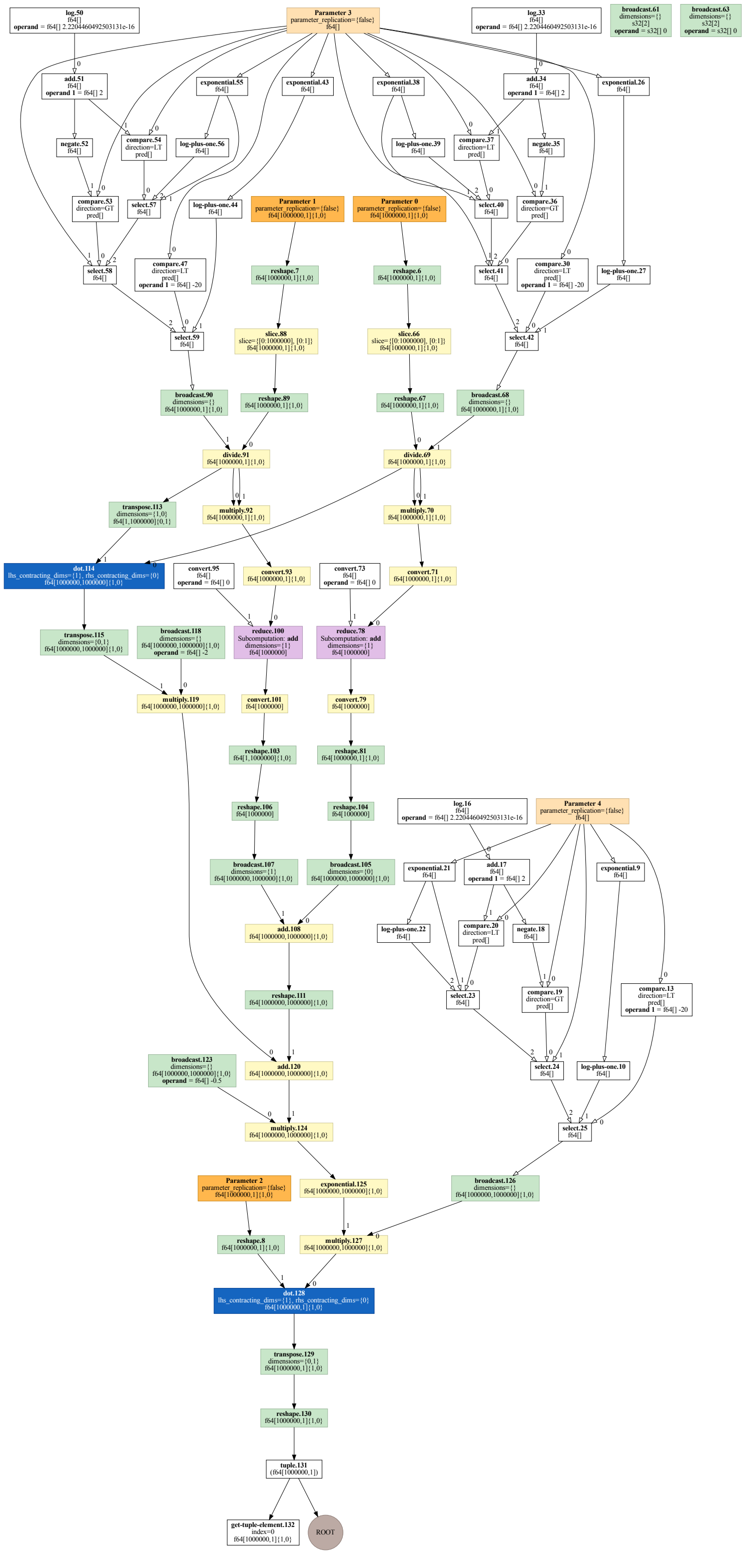

Figure 5: XLA HLO graph for kernel matrix-vector multiplication **before** splitting optimisation pass is applied in the XLA optimisation pipeline.

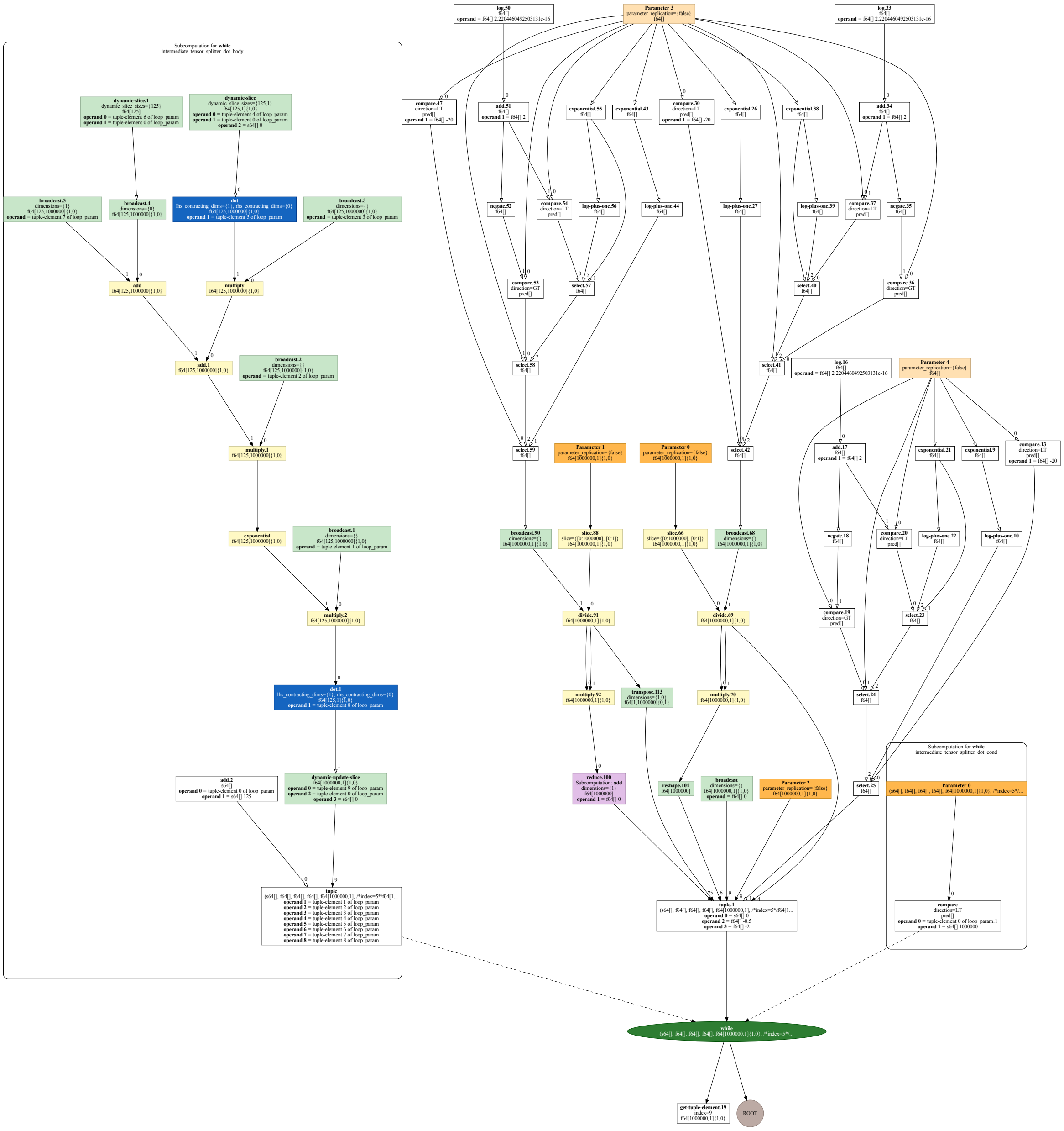
Subcomputation for while intermediate_tensor_splitter_dot_body
dynamic-slice.1 dynamic_slice_sizes={125} f64[125] operand 0 = tuple-element 6 of loop_param operand 1 = tuple-element 0 of loop_param
dynamic-slice dynamic_slice_sizes={125,1} f64[125,1]{1,0} operand 0 = tuple-element 4 of loop_param operand 1 = tuple-element 0 of loop_param operand 2 = s64[] 0
broadcast.5 dimensions={1} f64[125,1000000]{1,0} operand = tuple-element 7 of loop_param
broadcast.4 dimensions={0} f64[125,1000000]{1,0}
dot lhs_contracting_dims={1}, rhs_contracting_dims={0} f64[125,1000000]{1,0} operand 1 = tuple-element 5 of loop_param
broadcast.3 dimensions={} f64[125,1000000]{1,0} operand = tuple-element 3 of loop_param
add f64[125,1000000]{1,0}
multiply f64[125,1000000]{1,0}
add.1 f64[125,1000000]{1,0}
broadcast.2 dimensions={} f64[125,1000000]{1,0} operand = tuple-element 2 of loop_param
multiply.1 f64[125,1000000]{1,0}
exponential f64[125,1000000]{1,0}
broadcast.1 dimensions={} f64[125,1000000]{1,0} operand = tuple-element 1 of loop_param
multiply.2 f64[125,1000000]{1,0}
dot.1 lhs_contracting_dims={1}, rhs_contracting_dims={0} f64[125,1]{1,0} operand 1 = tuple-element 8 of loop_param
add.2 s64[] operand 0 = tuple-element 0 of loop_param operand 1 = s64[] 125
dynamic-update-slice f64[1000000,1]{1,0} operand 0 = tuple-element 9 of loop_param operand 2 = tuple-element 0 of loop_param operand 3 = s64[] 0
tuple (s64[], f64[], f64[], f64[], f64[1000000,1], /*index=5*/f64[1... operand 1 = tuple-element 1 of loop_param operand 2 = tuple-element 2 of loop_param operand 3 = tuple-element 3 of loop_param operand 4 = tuple-element 4 of loop_param operand 5 = tuple-element 5 of loop_param operand 6 = tuple-element 6 of loop_param operand 7 = tuple-element 7 of loop_param operand 8 = tuple-element 8 of loop_param
log.50 f64[] operand = f64[] 2.2204460492503131e-16
Parameter 3 parameter_replication={false} f64[]
log.33 f64[] operand = f64[] 2.2204460492503131e-16
compare.47 direction=LT pred[] operand 1 = f64[] -20
add.51 f64[] operand 1 = f64[] 2
exponential.55 f64[]
exponential.43 f64[]
compare.30 direction=LT pred[] operand 1 = f64[] -20
exponential.26 f64[]
exponential.38 f64[]
add.34 f64[] operand 1 = f64[] 2
negate.52 f64[]
compare.54 direction=LT pred[]
log-plus-one.56 f64[]
log-plus-one.44 f64[]
log-plus-one.27 f64[]
log-plus-one.39 f64[]
compare.37 direction=LT pred[]
negate.35 f64[]
compare.53 direction=GT pred[]
select.57 f64[]
select.40 f64[]
compare.36 direction=GT pred[]
select.58 f64[]
select.41 f64[]
log.16 f64[] operand = f64[] 2.2204460492503131e-16
Parameter 4 parameter_replication={false} f64[]
select.59 f64[]
Parameter 1 parameter_replication={false} f64[1000000,1]{1,0}
Parameter 0 parameter_replication={false} f64[1000000,1]{1,0}
select.42 f64[]
add.17 f64[] operand 1 = f64[] 2
exponential.21 f64[]
exponential.9 f64[]
compare.13 direction=LT pred[] operand 1 = f64[] -20
broadcast.90 dimensions={} f64[1000000,1]{1,0}
slice.88 slice={[0:1000000], [0:1]} f64[1000000,1]{1,0}
slice.66 slice={[0:1000000], [0:1]} f64[1000000,1]{1,0}
broadcast.68 dimensions={} f64[1000000,1]{1,0}
negate.18 f64[]
compare.20 direction=LT pred[]
log-plus-one.22 f64[]
log-plus-one.10 f64[]
divide.91 f64[1000000,1]{1,0}
divide.69 f64[1000000,1]{1,0}
compare.19 direction=GT pred[]
select.23 f64[]
multiply.92 f64[1000000,1]{1,0}
transpose.113 dimensions={1,0} f64[1,1000000]{0,1}
multiply.70 f64[1000000,1]{1,0}
select.24 f64[]
reduce.100 Subcomputation: add dimensions={1} f64[1000000] operand 1 = f64[] 0
reshape.104 f64[1000000]
broadcast dimensions={} f64[1000000,1]{1,0} operand = f64[] 0
Parameter 2 parameter_replication={false} f64[1000000,1]{1,0}
select.25 f64[]
Subcomputation for while intermediate_tensor_splitter_dot_cond
Parameter 0 (s64[], f64[], f64[], f64[], f64[1000000,1]{1,0}, /*index=5*/...
tuple.1 (s64[], f64[], f64[], f64[], f64[1000000,1], /*index=5*/f64[1... operand 0 = s64[] 0 operand 2 = f64[] -0.5 operand 3 = f64[] -2
compare direction=LT pred[] operand 0 = tuple-element 0 of loop_param.1 operand 1 = s64[] 1000000
while (s64[], f64[], f64[], f64[], f64[1000000,1]{1,0}, /*index=5*/...
get-tuple-element.19 index=9 f64[1000000,1]{1,0}
ROOT

Figure 6: XLA HLO graph for kernel matrix-vector multiplication **after** splitting optimisation pass is applied in the XLA optimisation pipeline.