Example Class 4 — JAX on accelerators and the roofline

C1 Research Computing — Michaelmas 2026
MPhil in Data Intensive Science and MPhil in Computational Biology

Exercises

Python control flow under jit

`lotka_volterra` is the right-hand side of a predator–prey model. With `order` equal to 2, the growth of the prey is logistic.

```
1 import jax, jax.numpy as jnp
2
3 def lotka_volterra(z, order):
4     prey, pred = z[0], z[1]
5     growth = 1.1 * prey
6     if order == 2:
7         growth = growth * (1 - prey / 50.0)
8     return jnp.array([growth - 0.4 * prey * pred,
9                       0.1 * prey * pred - 0.4 * pred])
```

With `z = jnp.array([10.0, 5.0])`, the first line below returns `[-11.2, 3.0]`, and the second raises `jax.errors.TracerBoolConversionError`: Python asks for the truth value of a traced array.

```
lotka_volterra(z, 2)
jax.jit(lotka_volterra)(z, 2)
```

(a) Give the line that raises the error and, as one bullet point, why the value tested there is traced.

(b) Give three ways to get `[-11.2, 3.0]` from a compiled function:

- with the keyword `static_argnums` of `jax.jit`;
- with a lambda or a `functools.partial` that fixes `order`;
- with one statement in place of lines 6–7, using `jnp.where(c, a, b)` or `jax.lax.cond(c, f_true, f_false, operand)`; both accept a traced `c`.

(c) Each version is called with `order` equal to 2, 2, 1 and 2; the lambda or partial version uses one jitted function per value. How many times does each version compile?

Shapes known at trace time

`largest` returns the `k` largest entries of a one-dimensional array, largest first.

```
def largest(x, k):
    return -jnp.sort(-x)[:k]
```

For `x = jnp.array([3.0, 9.0, 1.0, 7.0, 4.0])`, `jax.jit(largest)(x, 3)` raises

```
IndexError: Array slice indices must have static start/stop/step to be used with
    NumPy indexing syntax. Got slice(None, JitTracer(~int32[]), None) at position 0.
     ...
```

(a) Give, as one bullet point, why `k` must be static, and a call of `jax.jit` that returns `[9.0, 7.0, 4.0]`. Give also `jnp.sort(x)[-3:]`.

(b) For a symmetric `C`, `w, V = jnp.linalg.eigh(C)` returns the eigenvalues in ascending order, with `V[:, i]` the eigenvector of `w[i]`. `U, s, Vt = jnp.linalg.svd(A)` returns the singular values in descending order, with `U[:, i]` the left singular vector of `s[i]`.

- For `C = jnp.diag(jnp.array([2.0, 7.0, 1.0, 5.0]))`, give `w` and the index of the nonzero entry of `V[:, -1]`.
- Give expressions for the eigenvectors of the `k` smallest eigenvalues of `C`, as columns, smallest first, and for the `k` leading left singular vectors of `A`, leading first.

(c) How many times does `f` compile in these lines, run in one process?

```
f = jax.jit(largest, static_argnames="k")
x1, x2, x3 = jnp.ones(1000), jnp.ones(3000), jnp.arange(1000)  # x3 is int32
f(x1, 3); f(x1 + 1, 3); f(x2, 3); f(x1, 7); f(x2, 3); f(x3, 3)
```

A timing harness for compiled functions

`run_times` calls `fn(x)` $n+1$ times, waits for each result, and returns the $n+1$ times in ms, the first included.

```
1 import time
2
3 def run_times(fn, x, n=9):
4     times = []
5     for _ in range(n + 1):
6         start = time.perf_counter()
7         fn(x).block_until_ready()
8         times.append(1e3 * (time.perf_counter() - start))
9     return times
```

Suppose that, for one jitted `f` and one array `x` on one TPU chip, `run_times(f, x)` returns 312.0, 4.12, 4.10, 4.15, 4.09, 7.31, 4.11, 4.13, 4.08 and 4.14 ms.

(a) Give the mean of the ten times, and the mean and the median of the last nine. Which of the three would you report as the time of `f`? Give one bullet point against each of the other two.

(b) Line 7 is replaced by `y = fn(x)`, and `y.block_until_ready()` is added after line 8, so that each result is still waited for. Would the last nine times be larger or smaller? Give one bullet point on why.

A randomised range finder

A randomised range finder approximates the k leading left singular vectors of an $m \times n$ matrix A. This variant, with $\ell = k + 5$, samples the rows of A:

1. draw an $\ell \times m$ matrix Ω of independent standard normal entries, and form $Y = \Omega A$;
2. take the thin QR factorisation of Y^{T}; its factor Q has ℓ orthonormal columns that span those of Y^{T};

3. form $C = AQ$ and its thin SVD, $C = U_C \Sigma W^\mathsf{T}$;
4. keep U_k, the first k columns of U_C.

The rows of A then lie nearly in the span of the columns of Q, so $A \approx AQQ^\mathsf{T} = CQ^\mathsf{T}$. `scipy.linalg.subspace_angles(P, R)` returns the principal angles, in radians, between the column spans of `P` and `R`.

(a) Take $m = 6000$, $n = 2500$ and $k = 15$. Give the shapes of Ω, Y, Q, C, U_C and U_k.

(b) A test compares U_k with U_{ref}, the first k columns of `U` from `jnp.linalg.svd(A)`, for an A of rank 15. In one run, `jnp.allclose(Uk, Uref, atol=1e-3)` is `False`, while the largest principal angle between the two column spans is 1.9×10^{-6} rad. Give two bullet points: what the angle shows, and why `allclose` fails all the same.

(c) Give two 3×2 matrices with orthonormal columns, each column of one differing from the corresponding column of the other, whose principal angles are both 0.

Operations, bytes and intensity

For one kernel, F counts its floating-point operations, in FLOP, each multiplication or addition counting 1. D counts the bytes it moves, each input read once and each output written once, at 2 bytes per bf16 number and 4 per float32. Then $I = F/D$. A run of time t achieves $R = F/t$, or $R_\% = 100R/P_{\max}$ of the peak, and the roofline time is $t_{\text{roof}} = \max(F/P_{\max}, D/W_{\max})$.

One TPU v5e chip has $P_{\max} = 197$ TFLOP/s and $W_{\max} = 819$ GB/s, so $I^* = P_{\max}/W_{\max} \approx 240$ FLOP/byte. A kernel is memory-bound if $I < I^*$, and compute-bound otherwise. Take $P_{\max}$ as the peak of every kernel below.

A finite impulse response (FIR) filter with K float32 taps h_j turns N float32 samples x_i into N float32 outputs

$$y_i = \sum_{j=0}^{K-1} h_j \, x_{i-j}, \qquad x_i = 0 \text{ for } i < 0.$$

Take $N = 10^8$, and suppose that the sum, computed as large matrix products on the matrix unit of one v5e chip, takes

K	1	8	64	256	512	2048
t (ms)	1.22	1.21	1.22	1.22	1.24	2.50

(a) Give F, to leading order, and D in terms of N and K.

(b) Give I at $K = 1$, and say whether the filter is then memory-bound or compute-bound. Give t_{roof} at $K = 2048$.

(c) Give the first K of the table at which I exceeds I^*.

(d) Give, as one bullet point, why t stays near 1.2 ms from $K = 1$ to $K = 512$. Give $R_\%$ at $K = 2048$.

Matrix products, factorisations and speed-ups

The cost model and the chip are those of the previous exercise. The matrix unit of a v5e nears $P_{\max}$ only when nearly all the work of a kernel is large matrix multiplication.

The product $C = ST$ of two $n \times n$ matrices reads S and T in bf16 and writes C in float32, with $F = 2n^3$. Suppose that it takes 2.64 ms at $n = 6000$ and 20.8 ms at $n = 12000$, and that NumPy takes 1.8 s for the float32 product at $n = 12000$ on a laptop CPU.

The Cholesky factorisation $A = LL^{\mathsf{T}}$ of a symmetric positive-definite $n \times n$ float32 matrix, by `jnp.linalg.cholesky`, has $F \approx n^3/3$; it reads A and writes L as an $n \times n$ float32 array. Suppose that it takes 590 ms at $n = 12000$.

(a) Give D and I of the product in terms of n, the ratio of its two times, and the ratio of its two roofline times. By what factor does t_{roof} at $n = 12000$ grow when S has $2n$ rows?

(b) Give $R_{\%}$ of the product at $n = 12000$, and the speed-up of the chip over the laptop.

(c) Give $R_{\%}$ of the factorisation, and say whether its I makes it memory-bound or compute-bound. Give, as one bullet point, why its t is so far above its t_{roof}.

A roofline drawing

On log–log axes with labels and units, draw the roofline $R_{\text{roof}}(I) = \min(P_{\max}, I\,W_{\max})$ of the v5e chip, and mark the ridge point I^*. Add, as labelled points (I, R), the FIR filter at $K = 1$ and $K = 2048$, and the product ST and the Cholesky factorisation at $n = 12000$.