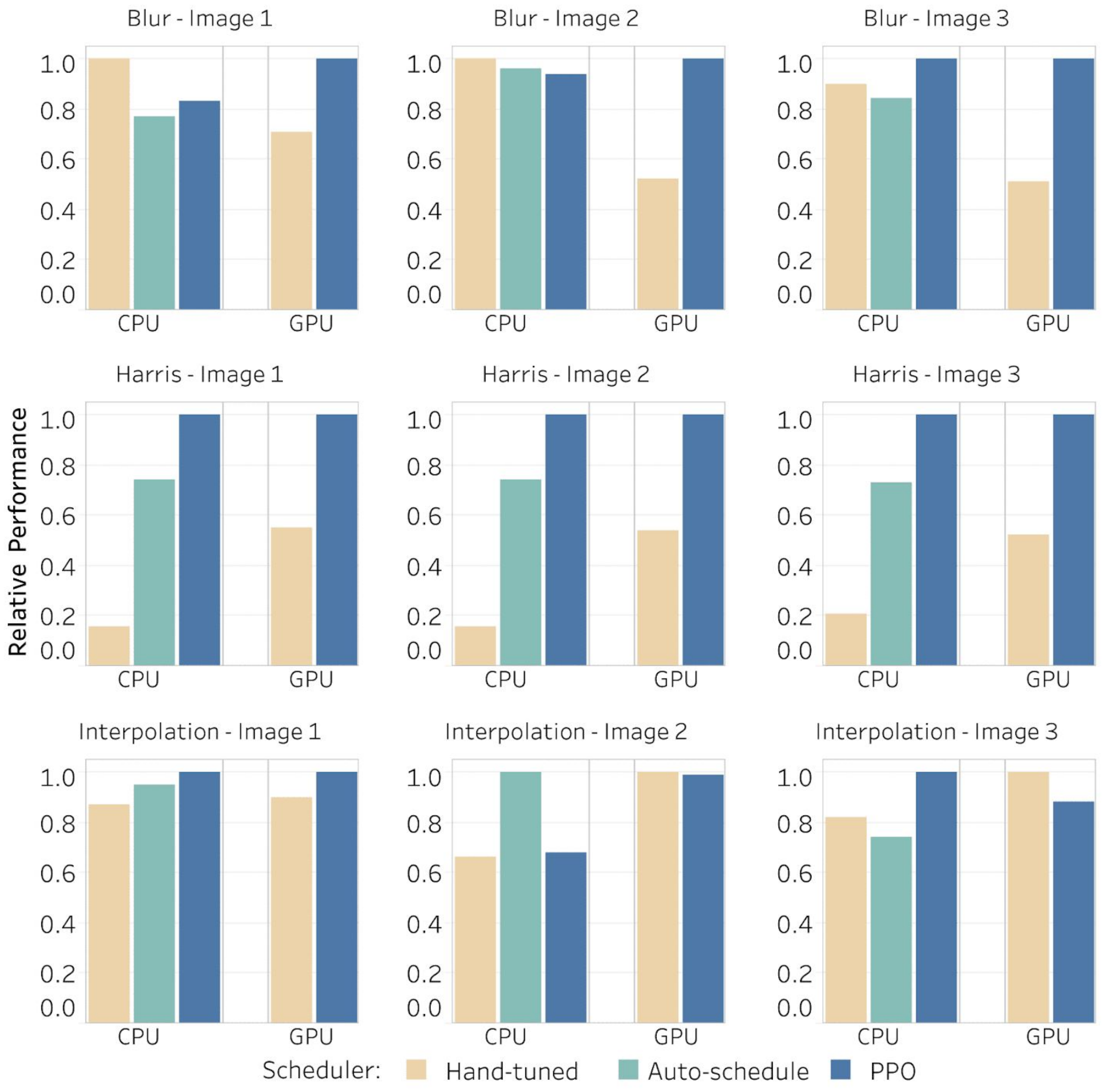
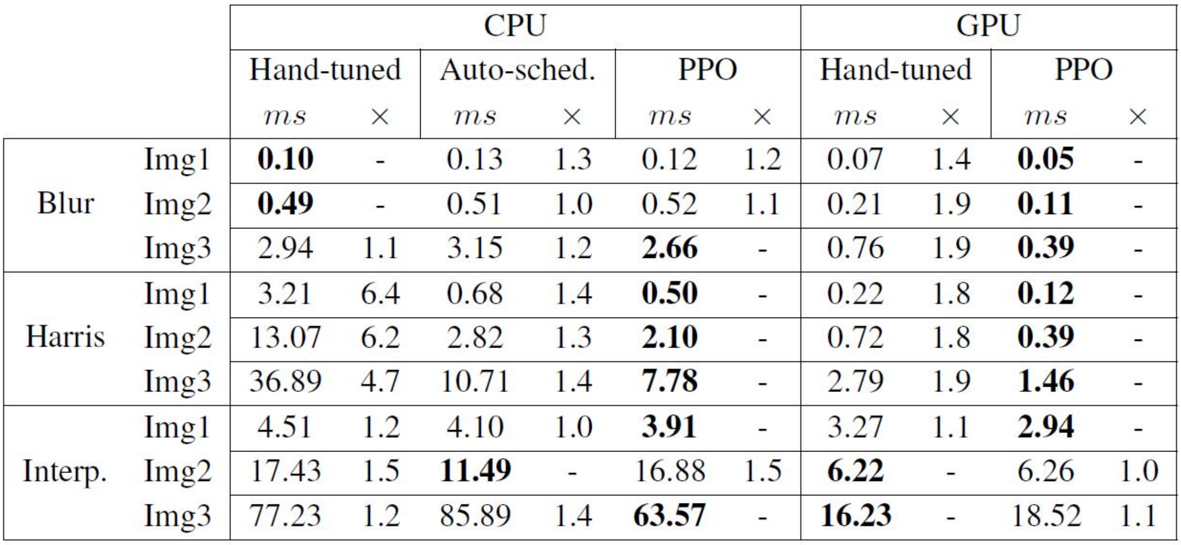
Optimization of Halide Image Processing Schedules with Reinforcement Learning

--- Generated Schedules Comparison ---

1. **Performance relative to the best result by architecture and scheduling method. The bigger the better. The relative performance is based on the ratio between the execution times of the compared schedules. The best has value 1.0 and the others have lower values.**



2. **Absolute Execution Time (ms) and Relative Slowdown (x) by Architecture and Scheduling Method.**



		CPU						GPU			
		Hand-tuned		Auto-sched.		PPO		Hand-tuned		PPO	
		ms	×	*ms*	×	*ms*	×	*ms*	×	*ms*	×
Blur	Img1	**0.10**	-	0.13	1.3	0.12	1.2	0.07	1.4	**0.05**	-
	Img2	**0.49**	-	0.51	1.0	0.52	1.1	0.21	1.9	**0.11**	-
	Img3	2.94	1.1	3.15	1.2	**2.66**	-	0.76	1.9	**0.39**	-
Harris	Img1	3.21	6.4	0.68	1.4	**0.50**	-	0.22	1.8	**0.12**	-
	Img2	13.07	6.2	2.82	1.3	**2.10**	-	0.72	1.8	**0.39**	-
	Img3	36.89	4.7	10.71	1.4	**7.78**	-	2.79	1.9	**1.46**	-
Interp.	Img1	4.51	1.2	4.10	1.0	**3.91**	-	3.27	1.1	**2.94**	-
	Img2	17.43	1.5	**11.49**	-	16.88	1.5	**6.22**	-	6.26	1.0
	Img3	77.23	1.2	85.89	1.4	**63.57**	-	**16.23**	-	18.52	1.1

3. **Best PPO Generated Schedule (CPU) for each Reinforcement Learning Trial:**

Blur	Trial 1	blur_y.tile(x, y, xi, yi, 512, 16);
		blur_y.parallel(y);
		blur_x.compute_at(blur_y, yi);
		blur_x.compute_at(blur_y, x);

	Trial 2	blur_y.unroll(x, 4);
		blur_y.parallel(y);
		blur_y.bound(x, 0, input.width());

	Trial 3	blur_y.tile(x, y, xi, yi, 128, 8);
		blur_y.unroll(x, 2);
		blur_x.compute_at(blur_y, x);
		blur_y.parallel(y);
		blur_y.bound(y, 0, input.height());
		blur_y.unroll(x, 4);

	Trial 4	blur_y.tile(x, y, xi, yi, 512, 16);
		blur_y.parallel(y);
		blur_x.compute_at(blur_y, x);

Harris	Trial 1	shifted.tile(x, y, xi, yi, 256, 64);
		shifted.parallel(y);
		gray.compute_at(shifted, x);
		Iy.compute_at(shifted, x);
		Ix.compute_at(shifted, x);

	Trial 2	shifted.tile(x, y, xi, yi, 512, 8);
		shifted.parallel(y);
		gray.compute_at(shifted, x);
		Iy.compute_at(shifted, x);
		Ix.compute_at(shifted, x);
		gray.store_at(shifted, y);
		gray.unroll(x, 2);
		gray.unroll(x, 4);

	Trial 3	shifted.tile(x, y, xi, yi, 512, 32);
		Ix.compute_at(shifted, x);
		shifted.parallel(y);
		gray.compute_at(shifted, x);
		Iy.compute_at(shifted, x);
		gray.unroll(x, 4);

	Trial 4	shifted.tile(x, y, xi, yi, 512, 32);
		shifted.parallel(y);
		gray.compute_at(shifted, x);
		Iy.compute_at(shifted, x);
		gray.unroll(x, 4);
		Ix.compute_at(shifted, x);

Interp.	Initial Setup	// Initial schedule setup needed to compile
		for (int l = 1; l < levels-1; ++l) {
		    downsampled[l].compute_root();
		    interpolated[l].compute_root();
		}

	Trial 1	downsampled[7].tile(x, y, xi, yi, 128, 8);
		normalize.parallel(y);
		downsampled[1].parallel(y);
		interpolated[1].parallel(c);
		interpolated[5].unroll(c, 3);
		normalize.vectorize(x, 16);
		downsampled[2].parallel(c);
		interpolated[2].parallel(y);
		downsampled[3].unroll(x, 2);
		interpolated[1].vectorize(x, 4);

	Trial 2	normalize.parallel(y);
		interpolated[1].vectorize(x, 4);
		downsampled[1].parallel(c);
		downsampled[1].unroll(x, 4);
		interpolated[1].parallel(c);
		normalize.vectorize(x, 16);
		downsampled[2].parallel(y);
		downsampled[5].vectorize(x, 8);
		interpolated[2].unroll(x, 4);

	Trial 3	interpolated[2].tile(x, y, xi, yi, 256, 32);

		normalize.parallel(y);
		downsampled[1].parallel(y);
		normalize.bound(x, 0, input.width());
		interpolated[1].parallel(c);
		interpolated[1].vectorize(x, 8);
		normalize.unroll(x, 4);
		downsampled[2].parallel(y);
		normalize.unroll(x, 2);
		downsampled[2].vectorize(x, 16);
		interpolated[2].parallel(c);
		downsampled[2].parallel(c);

	Trial 4	downsampled[6].tile(x, y, xi, yi, 16, 8);
		normalize.parallel(y);
		downsampled[1].parallel(c);
		normalize.vectorize(x, 16);
		interpolated[1].parallel(c);
		downsampled[2].tile(x, y, xi, yi, 64, 8);
		downsampled[1].unroll(x, 4);
		downsampled[2].parallel(y);
		interpolated[2].parallel(y);

4. Best PPO Generated Schedule (GPU) for each Reinforcement Learning Trial:

Blur	Trial 1	blur_y.gpu_tile(x, y, xi, yi, 64, 8);
		blur_y.unroll(yi, 4);
		blur_y.unroll(xi, 2);

	Trial 2	blur_y.gpu_tile(x, y, xi, yi, 128, 8);
		blur_y.unroll(yi, 4);
		blur_y.unroll(xi, 2);

	Trial 3	blur_y.gpu_tile(x, y, xi, yi, 64, 8);
		blur_y.unroll(xi, 2);
		blur_y.unroll(yi, 4);

	Trial 4	blur_y.gpu_tile(x, y, xi, yi, 64, 16);
		blur_y.unroll(yi, 4);
		blur_y.unroll(xi, 2);

Harris	Trial 1	shifted.gpu_tile(x, y, xi, yi, 64, 16);
		shifted.unroll(yi, 2);
		shifted.unroll(yi, 4);

	Trial 2	shifted.gpu_tile(x, y, xi, yi, 32, 16);
		shifted.unroll(yi, 4);
		shifted.unroll(yi, 2);

	Trial 3	shifted.compute_root();
		shifted.gpu_tile(x, y, xi, yi, 8, 32);
		shifted.bound(y, 0, input.height());
		shifted.unroll(yi, 2);
		gray.compute_at(shifted, x);
		gray.gpu_threads(x, y);
		gray.unroll(y, 2);
		gray.unroll(x, 3);

	Trial 4	shifted.gpu_tile(x, y, xi, yi, 32, 16); shifted.unroll(yi, 4);

Interp.	Initial Setup	// Initial schedule setup needed to compile
		for (int l = 1; l < levels-1; ++l) {
		    downsampled[l].compute_root();
		    interpolated[l].compute_root();
		}

	Trial 1	normalize.gpu_tile(x, y, c, xi, yi, ci, 8, 8, 3);
		downsampled[1].gpu_tile(x, y, c, xi, yi, ci, 8, 8, 4);
		interpolated[1].gpu_tile(x, y, c, xi, yi, ci, 16, 16, 4);
		downsampled[2].gpu_tile(x, y, c, xi, yi, ci, 8, 16, 4);
		downsampled[6].gpu_tile(x, y, c, xi, yi, ci, 16, 8, 4);
		interpolated[3].gpu_tile(x, y, c, xi, yi, ci, 8, 8, 4);
		downsampled[4].gpu_tile(x, y, c, xi, yi, ci, 16, 8, 4);
		downx[5].compute_at(downsampled[5], x);
		downsampled[8].gpu_tile(x, y, c, xi, yi, ci, 16, 16, 4);
		interpolated[1].unroll(yi, 2);
		interpolated[8].gpu_tile(x, y, c, xi, yi, ci, 16, 16, 4);
		interpolated[2].gpu_tile(x, y, c, xi, yi, ci, 32, 8, 4);
		interpolated[2].unroll(y, 3);
		downsampled[5].gpu_tile(x, y, c, xi, yi, ci, 8, 32, 4);
		downsampled[6].unroll(xi, 4);
		downsampled[2].unroll(ci, 4);
		downsampled[4].unroll(yi, 2);
		interpolated[2].unroll(xi, 3);
		downsampled[5].unroll(ci, 3);
		downsampled[3].gpu_tile(x, y, c, xi, yi, ci, 8, 16, 4);

	Trial 2	normalize.gpu_tile(x, y, c, xi, yi, ci, 32, 8, 3);
		downsampled[1].gpu_tile(x, y, c, xi, yi, ci, 8, 16, 4);
		interpolated[1].gpu_tile(x, y, c, xi, yi, ci, 8, 8, 4);
		downsampled[2].gpu_tile(x, y, c, xi, yi, ci, 8, 8, 4);
		interpolated[2].gpu_tile(x, y, c, xi, yi, ci, 16, 16, 4);
		normalize.unroll(ci, 3);
		downsampled[3].gpu_tile(x, y, c, xi, yi, ci, 8, 8, 4);

	Trial 3	normalize.gpu_tile(x, y, c, xi, yi, ci, 8, 32, 3);
		downsampled[1].gpu_tile(x, y, c, xi, yi, ci, 32, 8, 4);
		interpolated[1].gpu_tile(x, y, c, xi, yi, ci, 16, 8, 4);
		downsampled[2].gpu_tile(x, y, c, xi, yi, ci, 8, 8, 4);

		interpolated[2].gpu_tile(x, y, c, xi, yi, ci, 16, 16, 4);
		downsampled[3].gpu_tile(x, y, c, xi, yi, ci, 8, 32, 4);
		interpolated[5].unroll(x, 4);
		interpolated[3].unroll(x, 4);
		interpolated[6].unroll(y, 4);
		normalize.unroll(ci, 2);
		downsampled[1].unroll(xi, 2);

	Trial 4	normalize.gpu_tile(x, y, c, xi, yi, ci, 8, 16, 3);
		downsampled[1].gpu_tile(x, y, c, xi, yi, ci, 8, 8, 4);
		interpolated[1].gpu_tile(x, y, c, xi, yi, ci, 16, 8, 4);
		downsampled[2].gpu_tile(x, y, c, xi, yi, ci, 16, 8, 4);
		downsampled[3].gpu_tile(x, y, c, xi, yi, ci, 8, 8, 4);
		interpolated[6].unroll(c, 2);
		interpolated[5].unroll(y, 3);
		interpolated[2].gpu_tile(x, y, c, xi, yi, ci, 8, 16, 4);
		interpolated[8].unroll(c, 3);
		interpolated[3].gpu_tile(x, y, c, xi, yi, ci, 16, 16, 4);
		downsampled[4].gpu_tile(x, y, c, xi, yi, ci, 16, 8, 4);