TIGRE 3.3.1 and ASTRA 2.5: a speed comparison of the projection operators on one and several GPUs

Ander Biguri

Department of Applied Mathematics and Theoretical Physics (DAMTP), University of Cambridge

November 2026

TIGRE technical note. Accompanies TIGRE 3.3.1 (CERN/TIGRE pull requests #903–#907 and #911–#917).

Abstract

We compare the four projection operators of TIGRE 3.3.1 and ASTRA 2.5 — forward projection (TIGRE's Siddon and interpolated projectors against ASTRA's) and backprojection (TIGRE's matched and FDK-weighted backprojectors against ASTRA's) — on a desktop GPU (an RTX 4070, Windows) and on a server with four Quadro RTX 6000 (Linux), for cone-beam, parallel-beam and fan-beam (2D) geometries, with host (NumPy) and GPU (PyTorch) arrays, from 32^3 to 1536^3. For every case where ASTRA is faster we give the reason. On one GPU of the RTX 4070, TIGRE 3.3.1 is faster in 129 of 205 comparisons and ASTRA in 64, 51 of which are TIGRE's interpolated projector, which was not optimised in this release; without it, TIGRE is faster in 127 of 151 and ASTRA in 13. In 3D, the Siddon projector takes about half of ASTRA's time and the backprojectors 0.54–0.67 of it; in 2D the backprojectors take a fifth, and the Siddon projector 1.0–1.3 times ASTRA's time, where ASTRA's interpolating projector does less work per ray. On several GPUs, TIGRE gains less than ASTRA from extra GPUs for problems that fit in one: its multi-GPU calls copy the whole image to every GPU (measured on one four-GPU server; other machines may differ). For the large problems TIGRE's multi-GPU design is for, its forward projection of a 1536^3 image with 1536 angles takes 20.5 s on one GPU and 10.4 s on four, against ASTRA's 83.8 and 24.2 s.

1 Introduction

TIGRE [1, 2, 3] and ASTRA [4] are the two open GPU toolboxes most used for tomographic reconstruction from Python. Their operators are often compared, and the answer depends on the problem: on the size of the image, on the geometry, on whether the data live on the host or on the GPU, and on how many GPUs share the work. A comparison made for the release of TIGRE 3.3.0 measured large cone-beam problems; this note covers the whole range, and it was made with a question in mind: where TIGRE is slower, why?

Answering it led to the changes in TIGRE 3.3.1 (Section 2). Most of them concern what a call does besides its kernels: allocating and freeing memory, waiting for the GPU, page-locking host memory, copying the geometry from Python. Those costs do not grow with the problem, so they decide the speed of small problems, of 2D (fan-beam) problems, of iterative algorithms, which call the operators hundreds of times, and of the operators inside a neural network. A 2D fan-beam projection of a 256^2 image with 360 angles took about 3 ms in TIGRE 3.3.0, of which 0.2 ms were kernels.

Section 3 describes the machines and the method, Section 4 the results, and Section 5 explains each case where ASTRA is faster.

2 What changed in TIGRE 3.3.1

Table 1: Changes in TIGRE 3.3.1 that affect speed. Results are bit-identical to 3.3.0's except where a sum is ordered differently (parallel-beam backprojection, 2×10^{-6} relative; fan-beam FDK filtering on some CPUs, 3×10^{-7}).

pull request	change
#904	Launch shapes for thin problems. Forward projectors: thread blocks along the long side of a detector of fewer than 9 rows (a one-row fan-beam detector left 8 of 9 threads idle), and several blocks of angles per launch. Cone backprojectors: the column of voxels per thread matched to images of fewer than 8 slices.
#905, #917	One rule for page-locking the caller's arrays: only when the image is split or more than two GPUs are used. On one GPU, and on two, locking an array cost more than it saved.
#907	The fixed cost of a call: the geometry copied from Python through typed memoryviews; the projection buffers of the memory check used instead of allocated again; buffers cleared and textures filled on the kernels' stream instead of waiting for the GPU; GPU arrays written to directly by the forward projectors; all projections in one texture on thin detectors; the per-kernel constants in ordinary host memory.
#911	Cone backprojectors: no wait for the GPU between the kernels of a chunk of projections.
#912	Backprojectors read the projections from a layered 2D texture (bilinear fetch) instead of a 3D texture (trilinear fetch at an exact projection index: the same value at half the texture rate). The parallel-beam kernel gets the per-projection set-up folded on the host, as the cone kernels already had.
#913, #916	Siddon forward projectors: several steps of the ray walk per pass, with selects instead of branches, so that several texture fetches are in flight at once (the same sums): four for the usual detector, fewer when the rays are far apart in a large image, where more fetches in flight evict each other from the cache (#916); and the constants of the next kernel launch computed on the host while the current kernels run, instead of after waiting for them.
#914	FDK: detectors with few rows (fan beam), or a multiple of 16 rows that fits a block twice, filtered several projections per FFT call; the cosine weights applied while the filtering copies each block into its FFT buffer, instead of in a pass of their own.
#915	SIRT, SART, OS-SART, CGLS: vector updates in the arrays the operators return, without full-size temporaries. The same arithmetic.
#903, #906	Fixes: a crash in `Atb` at 1024^3 with 32-bit sizes; page-locking that fails falls back to pageable memory; a crash of the backprojection on 4+ GPUs when a GPU gets no slices.

FDK and the iterative algorithms (#914, #915) are not compared in this note, which covers the operators only.

3 Setup

Machines. Windows 11 (build 26300), NVIDIA GeForce RTX 4070 (12 GB, compute capability 8.9), 32 GB RAM, Python 3.11, ASTRA 2.5.0. Linux server (Ubuntu 22.04, kernel 5.15): 4 NVIDIA Quadro RTX 6000 (24 GB each, compute capability 7.5), 126 GB RAM, CUDA

12.5, Python 3.12, PyTorch 2.6, ASTRA 2.5.0, TIGRE built for sm_75. Both toolboxes are used through their Python interfaces: TIGRE's `Ax` and `Atb`, and ASTRA's `direct_FP3D` and `direct_BP3D` from `astra.experimental` (in 2D, `direct_FP2D` and `direct_BP2D`), which take host arrays or GPU tensors without copies on the Python side. Only the operators are compared: reconstruction algorithms (FDK, SIRT, CGLS) depend on more than the operators (filtering, vector arithmetic, where the data live) and are left out.

Same problems. The toolboxes use different conventions for angles, detector orientation and offsets. For each beam type the suite searches the ASTRA convention that reproduces TIGRE's projections on a small problem and uses it throughout (Table 2), so that both compute the same projections of the same image: their results agree to 5×10^{-4} (relative ℓ_2; TIGRE's Siddon projector integrates exactly, ASTRA's interpolates).

Timing. Each measurement is the median of several calls, repeated until it lasts at least 0.3 s; the two toolboxes alternate in rounds (three, the order changing every round), and the result is the ratio $t_{\mathrm{TIGRE}}/t_{\mathrm{ASTRA}}$ with a 95% confidence interval over the rounds. Below 1, TIGRE is faster (green); above, ASTRA (red); † marks an interval that contains 1. Host arrays include the copies to and from the GPU; GPU arrays (PyTorch tensors) do not, for either toolbox. Changes to TIGRE were measured the same way, alternating the two builds and using ASTRA, run in between, as a control for the machine's drift.

Table 2: Windows, RTX 4070. Calibration: the ASTRA convention that matches TIGRE, found on a small problem for each beam type, and how well the projections agree under it (relative L_2 difference after a least-squares scale). "Next best": the best convention that gives measurably different projections.

beam	ASTRA convention	agree	scale	next best
cone	$\theta_A = +1\,\theta_T + 90°$	5.4e-04	0.9998	1.6e-01
parallel	$\theta_A = +1\,\theta_T + 270°$, u flipped	5.1e-04	0.9998	4.9e-01
fan	$\theta_A = -1\,\theta_T + 90°$, u flipped	5.4e-04	0.9999	1.2e-01
offset	detector shift $+1\,u, +1\,v$	5.2e-04	0.9998	3.9e-01

4 Results

4.1 One GPU

Tables 3 (RTX 4070) and 4 (Quadro RTX 6000) summarise the comparisons on one GPU; the tables that follow give every case on the RTX 4070, and Appendix A those on the Quadro.

Table 3: Summary, one GPU (RTX 4070, Windows): median ratio $t_{\mathrm{TIGRE}}/t_{\mathrm{ASTRA}}$ over the cases of each class (below 1: TIGRE faster), its range, and the median speed-up of TIGRE 3.3.1 over 3.3.0 on the same cases ("–": not measured with 3.3.0).

operation	problems	arrays	cases	median	range	vs 3.3.0
fp Siddon	small 3D ($\leq 128^3$)	NumPy	4	0.63	0.50–0.69	1.61×
fp Siddon	small 3D ($\leq 128^3$)	tensors	4	0.65	0.53–0.92	2.30×
fp Siddon	cone 256^3–1024^3	NumPy	15	0.46	0.29–0.81	1.10×
fp Siddon	cone 256^3–1024^3	tensors	15	0.49	0.06–0.95	1.21×
fp Siddon	parallel	NumPy	3	0.59	0.55–0.63	–
fp Siddon	parallel	tensors	3	0.75	0.10–0.86	–
fp Siddon	fan (2D)	NumPy	5	1.17	1.11–1.22	4.87×
fp Siddon	fan (2D)	tensors	5	1.17	1.02–1.32	6.57×
fp interpolated	small 3D ($\leq 128^3$)	NumPy	4	1.18	0.84–2.38	1.25×
fp interpolated	small 3D ($\leq 128^3$)	tensors	4	1.68	1.06–4.68	1.43×
fp interpolated	cone 256^3–1024^3	NumPy	15	2.64	1.85–3.09	0.97×
fp interpolated	cone 256^3–1024^3	tensors	15	3.31	0.83–4.56	0.98×
fp interpolated	parallel	NumPy	3	3.04	2.89–3.10	–
fp interpolated	parallel	tensors	3	4.44	1.08–4.89	–
fp interpolated	fan (2D)	NumPy	5	5.76	3.20–6.79	2.32×
fp interpolated	fan (2D)	tensors	5	7.23	3.05–8.17	2.46×
bp matched	small 3D ($\leq 128^3$)	NumPy	4	0.85	0.55–0.92	1.75×
bp matched	small 3D ($\leq 128^3$)	tensors	4	1.23	0.73–1.71	1.87×
bp matched	cone 256^3–1024^3	NumPy	15	0.65	0.50–0.90	1.37×
bp matched	cone 256^3–1024^3	tensors	13	0.67	0.54–0.77	1.47×
bp matched	fan (2D)	NumPy	5	0.21	0.19–0.43	5.88×
bp matched	fan (2D)	tensors	5	0.18	0.15–0.36	7.22×
bp FDK weights	small 3D ($\leq 128^3$)	NumPy	4	0.87	0.56–1.02	1.71×
bp FDK weights	small 3D ($\leq 128^3$)	tensors	4	1.29	0.69–1.66	1.84×
bp FDK weights	cone 256^3–1024^3	NumPy	15	0.59	0.45–0.88	1.52×
bp FDK weights	cone 256^3–1024^3	tensors	13	0.54	0.51–0.65	1.64×
bp FDK weights	fan (2D)	NumPy	5	0.19	0.18–0.42	6.16×
bp FDK weights	fan (2D)	tensors	5	0.16	0.13–0.34	7.80×
bp	parallel	NumPy	3	0.55	0.49–0.61	–
bp	parallel	tensors	2	0.53	0.52–0.53	–

Table 4: Summary, one GPU (Quadro RTX 6000, Linux): median ratio $t_{\mathrm{TIGRE}}/t_{\mathrm{ASTRA}}$ over the cases of each class (below 1: TIGRE faster), its range, and the median speed-up of TIGRE 3.3.1 over 3.3.0 on the same cases ("–": not measured with 3.3.0).

operation	problems	arrays	cases	median	range	vs 3.3.0
fp Siddon	small 3D ($\leq 128^3$)	NumPy	4	0.61	0.54–0.69	1.75×
fp Siddon	small 3D ($\leq 128^3$)	tensors	4	0.70	0.52–0.83	2.29×
fp Siddon	cone 256^3–1024^3	NumPy	15	0.48	0.40–0.81	1.74×
fp Siddon	cone 256^3–1024^3	tensors	15	0.49	0.42–0.90	2.17×
fp Siddon	parallel	NumPy	3	0.66	0.60–0.94	1.81×
fp Siddon	parallel	tensors	3	0.70	0.69–0.84	2.05×
fp Siddon	fan (2D)	NumPy	5	1.67	1.47–2.25	8.41×
fp Siddon	fan (2D)	tensors	5	1.59	1.49–2.20	10.01×
fp interpolated	small 3D ($\leq 128^3$)	NumPy	4	1.03	0.80–2.06	1.36×
fp interpolated	small 3D ($\leq 128^3$)	tensors	4	1.30	1.01–3.18	1.55×
fp interpolated	cone 256^3–1024^3	NumPy	15	2.60	1.78–3.13	1.00×
fp interpolated	cone 256^3–1024^3	tensors	15	3.34	2.78–3.79	1.01×
fp interpolated	parallel	NumPy	3	2.96	2.66–3.27	1.01×
fp interpolated	parallel	tensors	3	4.17	4.12–4.72	1.01×
fp interpolated	fan (2D)	NumPy	5	6.04	4.18–7.85	2.50×
fp interpolated	fan (2D)	tensors	5	7.41	4.31–8.30	2.54×
bp matched	small 3D ($\leq 128^3$)	NumPy	4	0.88	0.72–0.90	1.70×
bp matched	small 3D ($\leq 128^3$)	tensors	4	1.10	0.91–1.13	1.79×
bp matched	cone 256^3–1024^3	NumPy	15	1.08	0.64–1.75	1.39×
bp matched	cone 256^3–1024^3	tensors	15	0.80	0.76–0.92	1.51×
bp matched	fan (2D)	NumPy	5	0.33	0.25–0.74	5.61×
bp matched	fan (2D)	tensors	5	0.22	0.19–0.63	7.69×
bp FDK weights	small 3D ($\leq 128^3$)	NumPy	4	0.87	0.71–0.90	1.69×
bp FDK weights	small 3D ($\leq 128^3$)	tensors	4	1.04	0.79–1.12	1.83×
bp FDK weights	cone 256^3–1024^3	NumPy	15	0.98	0.67–1.73	1.40×
bp FDK weights	cone 256^3–1024^3	tensors	15	0.60	0.59–0.77	1.60×
bp FDK weights	fan (2D)	NumPy	5	0.31	0.24–0.73	5.13×
bp FDK weights	fan (2D)	tensors	5	0.20	0.17–0.62	6.77×
bp	parallel	NumPy	3	0.98	0.87–0.98	1.66×
bp	parallel	tensors	3	0.54	0.54–0.57	2.46×

Table 5: Windows, RTX 4070. Cone beam, n^3 image, n^2 detector covering the image (409.6 mm), 360 angles, 1 GPU.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
128^3	fp, Siddon	7.90	11.5	0.68 ± 0.09	5.35	5.81	$0.93 \pm 0.10^{\dagger}$
	fp, interpolated	27.3	11.5	2.36 ± 0.06	27.2	5.81	4.65 ± 0.20
	bp, matched	5.21	9.44	0.56 ± 0.06	3.52	4.80	0.76 ± 0.13
	bp, FDK weights	5.25	9.44	0.57 ± 0.11	3.31	4.80	0.70 ± 0.07
256^3	fp, Siddon	32.1	63.9	0.50 ± 0.05	27.1	38.7	0.70 ± 0.05
	fp, interpolated	181	63.9	2.86 ± 0.23	177	38.7	4.55 ± 0.07
	bp, matched	32.5	54.7	0.58 ± 0.05	21.3	33.0	0.65 ± 0.03
	bp, FDK weights	29.1	54.7	0.53 ± 0.01	18.4	33.0	0.56 ± 0.02
512^3	fp, Siddon	241	526	0.46 ± 0.01	202	407	0.50 ± 0.01
	fp, interpolated	1384	526	2.63 ± 0.08	1366	407	3.35 ± 0.06
	bp, matched	242	375	0.65 ± 0.01	163	253	0.64 ± 0.01
	bp, FDK weights	219	375	0.59 ± 0.03	136	253	0.54 ± 0.02
1024^3	fp, Siddon	2137	4549	0.48 ± 0.04	1697	24480	0.07 ± 0.01
	fp, interpolated	11280	4549	2.48 ± 0.02	20439	24480	0.86 ± 0.10
	bp, matched	2076	2964	0.73 ± 0.18	—	—	—
	bp, FDK weights	1825	2964	0.64 ± 0.07	—	—	—

Table 6: Windows, RTX 4070. The 3.3.0 note's detector (0.8 mm pixels whatever n), 1 GPU. Compare with Table 5 at the same n.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
256^3, 0.8 mm	fp, Siddon	31.5	56.8	0.55 ± 0.05	27.4	37.3	0.73 ± 0.04
	fp, interpolated	175	56.8	3.08 ± 0.17	170	37.3	4.57 ± 0.14
	bp, matched	31.9	54.9	0.58 ± 0.04	21.3	32.6	0.64 ± 0.03
	bp, FDK weights	28.9	54.9	0.53 ± 0.08	18.0	32.6	0.56 ± 0.01
1024^3, 0.8 mm	fp, Siddon	1352	4276	0.32 ± 0.02	1013	17949	0.06 ± 0.00
	fp, interpolated	12300	4276	2.87 ± 0.07	22654	17949	1.27 ± 0.05
	bp, matched	1959	2655	0.74 ± 0.03	—	—	—
	bp, FDK weights	1739	2655	0.67 ± 0.05	—	—	—

Table 7: Windows, RTX 4070. Cone beam 512^3, varying the number of angles, 1 GPU.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
45 angles	fp, Siddon	82.5	119	0.69 ± 0.10	43.2	55.6	0.79 ± 0.07
	fp, interpolated	219	119	1.83 ± 0.25	181	55.6	3.27 ± 0.04
	bp, matched	111	123	0.91 ± 0.06	33.0	42.8	0.77 ± 0.02
	bp, FDK weights	108	123	$0.91 \pm 0.20^{\dagger}$	27.9	42.8	0.66 ± 0.01
90 angles	fp, Siddon	105	162	0.67 ± 0.22	67.5	93.5	0.72 ± 0.03
	fp, interpolated	380	162	2.30 ± 0.22	344	93.5	3.69 ± 0.07
	bp, matched	128	154	$0.87 \pm 0.29^{\dagger}$	50.6	70.3	0.72 ± 0.01
	bp, FDK weights	120	154	0.78 ± 0.12	42.8	70.3	0.60 ± 0.01
180 angles	fp, Siddon	147	284	0.52 ± 0.04	111	198	0.56 ± 0.02
	fp, interpolated	708	284	2.49 ± 0.01	672	198	3.39 ± 0.04
	bp, matched	181	227	0.77 ± 0.09	88.6	129	0.69 ± 0.02
	bp, FDK weights	150	227	0.65 ± 0.05	72.9	129	0.57 ± 0.00
360 angles	fp, Siddon	232	519	0.45 ± 0.01	198	400	0.50 ± 0.02
	fp, interpolated	1372	519	2.64 ± 0.11	1324	400	3.31 ± 0.01
	bp, matched	243	364	0.67 ± 0.10	165	247	0.66 ± 0.02
	bp, FDK weights	217	364	0.59 ± 0.02	134	247	0.54 ± 0.02
720 angles	fp, Siddon	401	944	0.43 ± 0.02	374	810	0.47 ± 0.01
	fp, interpolated	2655	944	2.81 ± 0.06	2635	810	3.25 ± 0.01
	bp, matched	411	632	0.64 ± 0.04	321	481	0.66 ± 0.02
	bp, FDK weights	333	632	0.53 ± 0.03	255	481	0.53 ± 0.01
1440 angles	fp, Siddon	755	1861	0.40 ± 0.01	731	1531	0.48 ± 0.01
	fp, interpolated	5290	1861	2.83 ± 0.04	5280	1531	3.44 ± 0.04
	bp, matched	721	1223	0.59 ± 0.01	644	950	0.68 ± 0.01
	bp, FDK weights	599	1223	0.48 ± 0.02	501	950	0.53 ± 0.01

Table 8: Windows, RTX 4070. Non-cubic problems, 1 GPU: clinical CBCT ($256 \times 512 \times 512$ image, 512×1024 detector), fine detector (256^3 image, 1024^2 detector), coarse detector (512^3 image, 256^2 detector), 360 angles.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
clinical CBCT	fp, Siddon	298	896	0.33 ± 0.01	288	774	0.37 ± 0.01
	fp, interpolated	2532	896	2.81 ± 0.09	2521	774	3.26 ± 0.02
	bp, matched	139	276	0.52 ± 0.12	93.9	146	0.64 ± 0.00
	bp, FDK weights	126	276	0.45 ± 0.02	78.0	146	0.54 ± 0.01
fine detector	fp, Siddon	302	1035	0.30 ± 0.06	306	773	0.40 ± 0.02
	fp, interpolated	2408	1035	2.32 ± 0.22	2419	773	3.13 ± 0.01
	bp, matched	138	257	0.54 ± 0.02	42.6	79.4	0.54 ± 0.01
	bp, FDK weights	144	257	0.56 ± 0.23	40.4	79.4	0.51 ± 0.01
coarse detector	fp, Siddon	142	174	0.82 ± 0.05	101	106	0.95 ± 0.01
	fp, interpolated	445	174	2.54 ± 0.10	407	106	3.82 ± 0.05
	bp, matched	233	322	0.72 ± 0.07	161	234	0.69 ± 0.05
	bp, FDK weights	222	322	0.68 ± 0.11	132	234	0.57 ± 0.03

Table 9: Windows, RTX 4070. Cone beam 512^3 with the detector shifted sideways by 102.4 mm (a quarter of its width; ASTRA `cone_vec`), 1 GPU.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
u offset 102.4 mm	fp, Siddon	203	507	0.39 ± 0.02	167	398	0.42 ± 0.00
	fp, interpolated	1349	507	2.65 ± 0.06	1318	398	3.31 ± 0.01
	bp, matched	243	381	0.66 ± 0.20	162	246	0.66 ± 0.01
	bp, FDK weights	222	381	0.58 ± 0.06	133	246	0.54 ± 0.00

Table 10: Windows, RTX 4070. Parallel beam (TIGRE parallel mode, ASTRA `parallel3d`), n^3, 360 angles, 1 GPU.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
256^3	fp, Siddon	30.1	54.8	0.55 ± 0.02	27.0	31.4	0.85 ± 0.03
	fp, interpolated	158	54.8	2.90 ± 0.06	154	31.4	4.87 ± 0.06
	bp	27.7	56.4	0.49 ± 0.05	16.9	32.2	0.53 ± 0.00
512^3	fp, Siddon	230	388	0.60 ± 0.08	196	263	0.74 ± 0.01
	fp, interpolated	1202	388	3.12 ± 0.11	1166	263	4.43 ± 0.03
	bp	200	362	0.57 ± 0.12	124	236	0.53 ± 0.01
1024^3	fp, Siddon	1965	3120	0.64 ± 0.03	1601	16762	0.10 ± 0.00
	fp, interpolated	9482	3120	3.03 ± 0.12	18039	16762	1.08 ± 0.00
	bp	1583	2590	0.64 ± 0.12	—	—	—

Table 11: Windows, RTX 4070. Fan beam (2D): TIGRE's one-slice cone geometry against ASTRA's `fanflat` (FP_CUDA, BP_CUDA, FBP_CUDA), n^2 image, n detector pixels, 1 GPU.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
512^2	fp, Siddon	1.51	1.28	1.17 ± 0.02	1.23	1.21	$1.09 \pm 0.16^{\dagger}$
	fp, interpolated	4.11	1.28	3.31 ± 0.69	3.68	1.21	3.15 ± 0.43
	bp, matched	1.56	3.62	0.43 ± 0.03	1.22	3.41	0.36 ± 0.00
	bp, FDK weights	1.51	3.62	0.44 ± 0.09	1.16	3.41	0.34 ± 0.00
1024^2	fp, Siddon	2.99	2.70	1.10 ± 0.03	2.15	2.09	1.04 ± 0.03
	fp, interpolated	11.3	2.70	4.17 ± 0.09	10.5	2.09	5.04 ± 0.14
	bp, matched	3.25	12.1	0.27 ± 0.01	2.42	11.5	0.21 ± 0.00
	bp, FDK weights	3.11	12.1	0.26 ± 0.02	2.25	11.5	0.20 ± 0.00
2048^2	fp, Siddon	7.69	6.90	1.12 ± 0.03	6.21	5.30	1.17 ± 0.03
	fp, interpolated	39.7	6.90	5.79 ± 0.24	38.3	5.30	7.22 ± 0.03
	bp, matched	9.64	46.6	0.21 ± 0.00	7.20	44.5	0.16 ± 0.00
	bp, FDK weights	8.96	46.6	0.19 ± 0.00	6.39	44.5	0.14 ± 0.00
4096^2	fp, Siddon	29.7	24.8	1.17 ± 0.11	24.2	18.8	1.29 ± 0.03
	fp, interpolated	158	24.8	6.30 ± 0.28	154	18.8	8.19 ± 0.13
	bp, matched	36.7	185	0.20 ± 0.01	26.7	175	0.15 ± 0.00
	bp, FDK weights	33.1	185	0.18 ± 0.01	23.1	175	0.13 ± 0.00
1024^2, 1440 angles	fp, Siddon	7.13	5.83	1.22 ± 0.04	6.67	5.05	1.32 ± 0.01
	fp, interpolated	39.6	5.83	6.78 ± 0.22	39.3	5.05	7.78 ± 0.05
	bp, matched	8.79	45.1	0.19 ± 0.00	7.94	44.5	0.18 ± 0.01
	bp, FDK weights	8.03	45.1	0.18 ± 0.00	7.16	44.5	0.16 ± 0.01

Table 12: Windows, RTX 4070. Small problems (cone beam, n^3 image, n^2 detector, 100 angles), 1 GPU.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
32^3	fp, Siddon	1.33	1.94	0.68 ± 0.05	0.97	1.26	$0.79 \pm 0.21^\dagger$
	fp, interpolated	1.64	1.94	0.86 ± 0.10	1.33	1.26	1.08 ± 0.05
	bp, matched	0.98	1.07	$0.94 \pm 0.14^\dagger$	0.94	0.55	1.64 ± 0.35
	bp, FDK weights	1.10	1.07	$1.00 \pm 0.21^\dagger$	0.91	0.55	1.58 ± 0.31
64^3	fp, Siddon	1.71	2.95	0.57 ± 0.09	1.01	1.91	0.53 ± 0.01
	fp, interpolated	2.75	2.95	$0.92 \pm 0.10^\dagger$	2.32	1.91	1.26 ± 0.23
	bp, matched	1.28	1.39	$0.97 \pm 0.22^\dagger$	0.92	0.62	1.51 ± 0.02
	bp, FDK weights	1.33	1.39	$0.95 \pm 0.08^\dagger$	1.00	0.62	1.58 ± 0.27
128^3	fp, Siddon	2.94	5.86	0.50 ± 0.01	1.88	3.56	0.53 ± 0.02
	fp, interpolated	8.36	5.86	1.44 ± 0.09	7.64	3.56	2.14 ± 0.02
	bp, matched	3.19	4.03	0.80 ± 0.07	1.63	1.68	$0.97 \pm 0.04^\dagger$
	bp, FDK weights	3.15	4.03	0.79 ± 0.03	1.61	1.68	$0.99 \pm 0.18^\dagger$

The fixed cost of a call. Table 13 follows one fan-beam call through the releases. In 3.3.0 a forward projection of a 256^2 image with 360 angles took 7.9 ms, of which 0.2 ms were kernels; the rest was allocating and freeing GPU and page-locked memory, waiting for the GPU, and converting the geometry in Python, all of which cost far more under Windows' WDDM driver model than the kernels.

Table 13: Wall time of one fan-beam call (256^2 image, 360 angles, PyTorch tensors, RTX 4070), from Nsight Systems traces, in ms.

version	Siddon fp	matched bp
3.3.0	7.9	–
+ launch shapes (#904)	3.0	3.0
+ fixed-cost changes (#907)	1.13	1.39
+ no wait between kernels (#911)	1.13	1.29
ASTRA 2.5	1.03	1.35

4.2 Several GPUs

Tables 14 and 15 give the times on 1, 2 and 4 GPUs of the Linux server, with NumPy arrays. Every problem in these tables fits in one 24 GB GPU. At 1024^3, two GPUs are faster than one for every TIGRE operator: the Siddon forward projection takes 1.79 s on one GPU and 1.44 s on two, faster than ASTRA on two (2.17 s), and the matched backprojection 3.03 and 2.36 s. At 512^3 two GPUs are faster for all but the Siddon projector, and at 256^3 only for the interpolated projector. On four GPUs TIGRE is slower than on two (at 1024^3, 2.08 s for the forward and 2.16 s for the backprojection), while ASTRA keeps gaining (1.68 and 0.96 s). TIGRE 3.3.0 took 3.37, 2.38 and 2.21 s for the same forward projection on 1, 2 and 4 GPUs: 3.3.1 is faster on every number of GPUs, but most on one, because its kernels are faster and the rest of a multi-GPU call is not.

That rest is a part of each multi-GPU call that does not shrink with the number of GPUs (Table 16): about 2 s at 1024^3 and 0.3–0.4 s at 512^3, almost the same for every TIGRE operator. We could not trace it on this server (it has no Nsight Systems); two parts of it are known. Every GPU needs a full set of the data it reads: a forward projection splits the angles between the GPUs, and every GPU gets the whole image (at 1024^3, 4 GB to each of them); a backprojection splits the image, and every GPU gets all the projections. And the caller's arrays are page-locked, so that the copies to the GPUs overlap. On two GPUs page-locking costs more than it saves, and

TIGRE 3.3.1 does not do it there (#917: 5–21% faster on two GPUs than with page-locking, 4–15% slower on four). The interpolated projector, whose kernels are long, is still 2.5 times faster on four GPUs than on one.

One machine. All the multi-GPU results in this note come from one server: four Quadro RTX 6000 (Turing, 2018) connected only through PCIe and the host (no NVLink; `nvidia-smi topo` reports `SYS` between every pair). A multi-GPU call depends on how fast the host feeds the GPUs, more than a one-GPU call does, so on other machines the balance between TIGRE and ASTRA, the number of GPUs from which TIGRE stops gaining, and the page-locking threshold of #917 may differ. We would welcome measurements on other multi-GPU machines, and on other GPUs in general: the suite that produced every table here is in this note's folder (Code availability), and results can be shared in an issue on TIGRE's GitHub.

Table 14: Linux, Quadro RTX 6000. Cone beam size sweep on 1, 2 and 4 GPUs (NumPy arrays). Speed-up: time on 1 GPU over time on g GPUs, for each toolbox on its own.

		TIGRE/ASTRA ratio			TIGRE speed-up		ASTRA speed-up	
problem	operation	1 GPU	2 GPUs	4 GPUs	2 GPUs	4 GPUs	2 GPUs	4 GPUs
128^3	fp, Siddon	0.64 ± 0.09	0.90 ± 0.09	1.40 ± 0.05	0.78	0.43	1.12	0.96
	fp, interpolated	2.06 ± 0.03	1.62 ± 0.45	1.66 ± 0.11	1.49	1.21	1.12	0.96
	bp, matched	0.72 ± 0.01	1.39 ± 0.04	2.18 ± 0.06	0.64	0.30	1.24	0.90
	bp, FDK weights	0.71 ± 0.01	1.39 ± 0.04	2.17 ± 0.11	0.63	0.30	1.24	0.90
256^3	fp, Siddon	0.75 ± 0.02	1.13 ± 0.08	1.68 ± 0.10	0.87	0.68	1.33	1.50
	fp, interpolated	2.29 ± 0.01	1.79 ± 0.12	1.79 ± 0.21	1.69	1.99	1.33	1.50
	bp, matched	0.99 ± 0.00	1.46 ± 0.06	2.49 ± 0.04	1.01	0.73	1.50	1.83
	bp, FDK weights	0.89 ± 0.01	1.45 ± 0.06	2.48 ± 0.04	0.91	0.66	1.50	1.83
512^3	fp, Siddon	0.46 ± 0.00	0.76 ± 0.03	1.53 ± 0.05	0.88	0.60	1.45	1.98
	fp, interpolated	2.60 ± 0.02	2.15 ± 0.04	2.22 ± 0.14	1.76	2.31	1.45	1.98
	bp, matched	1.09 ± 0.01	1.50 ± 0.26	2.50 ± 0.14	1.26	1.10	1.69	2.53
	bp, FDK weights	0.97 ± 0.01	1.44 ± 0.25	2.60 ± 0.15	1.17	0.95	1.69	2.53
1024^3	fp, Siddon	0.48 ± 0.00	0.66 ± 0.01	1.26 ± 0.12	1.24	0.86	1.73	2.24
	fp, interpolated	2.68 ± 0.02	2.65 ± 0.05	2.43 ± 0.21	1.75	2.50	1.73	2.24
	bp, matched	1.14 ± 0.01	1.37 ± 0.02	2.25 ± 0.06	1.28	1.41	1.55	2.78
	bp, FDK weights	0.99 ± 0.00	1.28 ± 0.02	2.15 ± 0.05	1.20	1.28	1.55	2.78

Table 15: Linux, Quadro RTX 6000. Parallel beam on 1, 2 and 4 GPUs (NumPy arrays).

		TIGRE/ASTRA ratio			TIGRE speed-up		ASTRA speed-up	
problem	operation	1 GPU	2 GPUs	4 GPUs	2 GPUs	4 GPUs	2 GPUs	4 GPUs
256^3	fp, Siddon	0.95 ± 0.02	1.44 ± 0.04	2.05 ± 0.09	0.84	0.63	1.29	1.36
	fp, interpolated	2.66 ± 0.01	2.04 ± 0.06	2.12 ± 0.05	1.68	1.72	1.29	1.36
	bp	0.87 ± 0.00	1.53 ± 0.06	2.65 ± 0.07	0.88	0.63	1.55	1.92
512^3	fp, Siddon	0.60 ± 0.00	$1.00 \pm 0.04^{\dagger}$	1.84 ± 0.07	0.86	0.59	1.46	1.80
	fp, interpolated	2.96 ± 0.03	2.45 ± 0.02	2.39 ± 0.18	1.76	2.25	1.46	1.80
	bp	0.98 ± 0.01	1.67 ± 0.10	2.79 ± 0.10	1.11	0.90	1.91	2.59
1024^3	fp, Siddon	0.65 ± 0.00	0.86 ± 0.02	1.56 ± 0.13	1.23	0.83	1.61	1.98
	fp, interpolated	3.26 ± 0.04	3.04 ± 0.05	2.70 ± 0.22	1.73	2.39	1.61	1.98
	bp	0.98 ± 0.02	1.28 ± 0.01	2.30 ± 0.01	1.17	1.20	1.54	2.81

Table 16: Linux, Quadro RTX 6000. Each call's time split into a part that does not shrink with more GPUs ("fixed") and a part that does, fitted as $t_g = \text{fixed} + \text{parallel}/g$ to the 1- and 4-GPU times (NumPy arrays). The measured 2-GPU time t_2 checks the fit.

problem	call	t_1 ms	t_4 ms	fixed ms	fixed %	t_2 ms	t_2 fit ms
cone 512^3	TIGRE fp Siddon	218	366	415	191	249	316
	TIGRE fp interpolated	1238	536	302	24	704	770
	TIGRE bp matched	373	338	326	87	297	350
	ASTRA fp	476	240	162	34	328	319
	ASTRA bp	344	136	67	19	204	206
cone 1024^3	TIGRE fp Siddon	1790	2075	2170	121	1441	1980
	TIGRE fp interpolated	10080	4039	2025	20	5772	6053
	TIGRE bp matched	3030	2155	1864	61	2358	2447
	TIGRE bp FDK	2638	2056	1861	71	2196	2250
	ASTRA fp	3754	1679	988	26	2170	2371
	ASTRA bp	2664	959	391	15	1719	1527
parallel 1024^3	TIGRE fp Siddon	1739	2099	2218	128	1418	1979
	TIGRE fp interpolated	8685	3628	1942	22	5016	5314
	TIGRE bp	2464	2053	1916	78	2100	2190
	ASTRA fp	2655	1339	900	34	1645	1778
	ASTRA bp	2512	893	353	14	1636	1432

4.3 Large problems: N^3 images, N angles

TIGRE's multi-GPU code was designed for a different regime [2]: as many angles as voxels along a side, which gives several times more kernel work for each byte moved, and images larger than one GPU, which TIGRE splits between the GPUs and through host memory. Table 17 measures it on the Linux server, with NumPy arrays. At 1536^3 the image takes 13.6 GB and the projections as much, more than one GPU's 24 GB together, and both toolboxes split the problem. TIGRE's Siddon forward projection is 4.1 times faster than ASTRA's on one GPU and 2.3 times on four, and its FDK-weighted backprojection 1.4 times faster on one GPU. Extra GPUs help ASTRA more: from one GPU to four its backprojection becomes 3.4 times faster and TIGRE's 2.0 times, so on four GPUs ASTRA's backprojection is faster (8.3 s, against 12.5 s for TIGRE's matched one).

Table 17: Linux, Quadro RTX 6000. Cone beam, N^3 image, N^2 detector covering the image, N angles, NumPy arrays: time of one call in s (median of three rounds). At 1536^3 the image and the projections do not fit in one GPU together, and both toolboxes split them.

N	GPUs	TIGRE fp Siddon	ASTRA fp	TIGRE bp matched	TIGRE bp FDK	ASTRA bp
512	1	0.29	0.66	0.44	0.39	0.45
	2	0.31	0.40	0.33	0.31	0.26
	4	0.43	0.29	0.39	0.41	0.17
1024	1	4.15	9.82	5.56	4.50	5.96
	2	2.67	5.58	3.62	3.14	3.37
	4	3.14	3.30	3.23	2.95	1.78
1536	1	20.50	83.82	25.57	20.14	28.51
	2	12.53	43.09	15.74	13.15	15.51
	4	10.35	24.16	12.53	11.10	8.34

5 Where ASTRA is faster, and why

On one GPU, every one of ASTRA's wins falls in one of four causes. On the RTX 4070 they are the interpolated projector (51 comparisons), the 2D Siddon projector (9) and the backprojection of the smallest images with GPU arrays (4). On the Quadro, the same three (52, 10 and 5) and the backprojection of large images from host arrays (14), whose cause we have narrowed down but not confirmed. On several GPUs, ASTRA gains more from extra GPUs, for the reasons given in the previous section.

The interpolated forward projector (1.0–8.3×). TIGRE's interpolated projector samples the image with trilinear 3D texture fetches every `geo.accuracy` = 0.5 voxels along the ray. ASTRA's projector (Joseph's method) takes one sample per slice crossed, with a bilinear fetch from a layered 2D texture. A trilinear fetch costs twice a bilinear one on NVIDIA GPUs (the effect that made TIGRE's backprojectors faster in 3.3.1, Section 2), and at the default step TIGRE takes about twice as many samples. Table 18 shows the cost of the step: at one sample per voxel TIGRE is still 1.7–4 times slower. A Joseph-type kernel would close the gap; it was not part of this release. TIGRE's Siddon projector, which computes exact intersection lengths, is the faster choice in 3D.

Table 18: TIGRE's interpolated forward projection against its sampling step (`geo.accuracy`, in voxels), fan beam, PyTorch tensors, ms.

problem	0.25	0.5 (default)	1	2	TIGRE Siddon	ASTRA
512^2, 360 angles	8.84	4.71	2.67	1.43	1.45	1.55
1024^2, 720 angles	55.9	28.8	15.2	5.07	4.81	3.69

The Siddon forward projector in 2D (1.0–1.3× on the RTX 4070, 1.5–2.4× on the Quadro). In 3D, TIGRE's Siddon projector takes 0.3–0.9 of ASTRA's time; in 2D fan beam it is slower. The kernel is: in Nsight Systems, at 1024^2 with 720 angles, TIGRE's kernels take 2.9 ms and ASTRA's 1.7 ms on the RTX 4070. Siddon's method visits every voxel boundary the ray crosses, about $1.3N$ steps for an $N \times N$ image against the N samples of ASTRA's 2D projector (Joseph's method), and each step chooses the next boundary. In 3D the same comparison favours TIGRE, because ASTRA's 3D projector pays for it in other ways (Section 5.5 of the Siddon note [5]). Reading the image through a layered 2D texture, the change that halved the backprojectors' cost, made the Siddon projector 1.3–1.5 times *slower*, and was not kept. The Quadro, an older GPU, is further behind ASTRA here.

Backprojection of very small images with GPU arrays (32^3 and 64^3). With GPU tensors, TIGRE's backprojection of a 32^3 image takes 0.94 ms and ASTRA's 0.55 ms on the RTX 4070. In an earlier run, with 0.93 and 0.62 ms, Nsight Systems shows 0.09 ms of kernels in both; TIGRE spends about 0.23 ms in Python before the call (checking and copying the geometry, kept on purpose: it catches wrong geometries before they reach the GPU) and about 0.2 ms more in CUDA calls than ASTRA (the projections' copy into the texture array and the per-kernel constants). From 128^3 TIGRE is faster.

Backprojection from host arrays on the Quadro (1.02–1.75×). On the Linux server, TIGRE's backprojection with GPU arrays takes 0.6–0.9 of ASTRA's time, but with NumPy arrays it is slower from 512^3, most with few angles (512^3, 45 angles: 240 ms against 137 ms). The difference is in what a call with host arrays adds to one with GPU arrays (Table 19): for TIGRE about 207 ms at 512^3 whatever the number of angles, for ASTRA 106 ms with 45 angles, growing with the projections it copies. On the RTX 4070 both add about 80 ms. TIGRE's added

time does not depend on the projections, so it is in the result: allocating the 512^3 image on the host (512 MB) and copying it back from the GPU. A likely cause is the host memory itself. TIGRE allocates the result with `malloc`, so the copy into it touches 131072 fresh 4 kB pages; ASTRA writes into an array NumPy allocated, which on Linux asks for huge pages. That would also explain why Windows does not show it. We have not yet confirmed it.

Table 19: What a call with NumPy arrays adds to the same call with GPU tensors: $t_{\text{host}} - t_{\text{tensors}}$ in ms, cone 512^3, one GPU.

GPU	angles	TIGRE bp matched	TIGRE bp FDK	ASTRA bp
Quadro RTX 6000	45	212	218	106
	180	206	209	115
	360	207	211	128
	1440	207	207	218
RTX 4070	45	79	80	81
	180	92	77	99
	360	77	83	117
	1440	76	98	273

6 Discussion

On one GPU, TIGRE 3.3.1's Siddon projector is faster than ASTRA's for every 3D problem, and both backprojectors for every 2D problem and every 3D one from 128^3 (256^3 on the Quadro). The exception is the Linux server, where the backprojectors are slower than ASTRA's with host arrays from 512^3 (Section 5). The changes that brought this were of two kinds. Most of 3.3.1's work removed fixed costs that did not grow with the problem and that had dominated small and 2D problems. The largest single change was not in the host code: reading the projections through a layered 2D texture instead of a 3D one halves the cost of every texture fetch in the backprojectors, without changing a bit of the results. ASTRA's remaining advantages come from an algorithmic choice (Joseph's interpolation against Siddon's exact lengths, in 2D), from the fixed cost of a call on the smallest images, or from the interpolated projector, which this release left as it was.

On several GPUs, TIGRE gains less than ASTRA from each extra GPU for problems that fit in one. Two GPUs are now faster than one for large images, but four are slower than two, because every multi-GPU call copies a full set of its input to every GPU. That part is the next to reduce, without giving up what TIGRE's multi-GPU path is for: splitting the image and the projections across GPUs and through host memory, so that it reconstructs images larger than one GPU, on one GPU or on several [2]. In that regime TIGRE's forward projection is 2.3–4.1 times faster than ASTRA's, its backprojection faster on one GPU and slower on four (Section 4.3). For a problem that fits in one GPU, one or two GPUs (`gpuids`) are the fastest choice in TIGRE 3.3.1 on the server we measured; with one multi-GPU machine, these conclusions need checking on others (Section 4, "One machine").

A Per-case results on the Linux server

One GPU of the Linux server (Quadro RTX 6000, Section 3); same suite and layout as the tables of Section 4.

Table 20: Linux, Quadro RTX 6000. Cone beam, n^3 image, n^2 detector covering the image (409.6 mm), 360 angles, 1 GPU.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
128^3	fp, Siddon	6.72	10.8	0.64 ± 0.09	4.96	6.73	0.78 ± 0.18
	fp, interpolated	22.3	10.8	2.06 ± 0.03	21.4	6.73	3.17 ± 0.05
	bp, matched	6.16	8.56	0.72 ± 0.01	4.31	4.75	0.91 ± 0.00
	bp, FDK weights	6.07	8.56	0.71 ± 0.01	3.74	4.75	0.79 ± 0.00
256^3	fp, Siddon	51.4	69.0	0.75 ± 0.02	23.6	44.2	0.53 ± 0.00
	fp, interpolated	158	69.0	2.29 ± 0.01	154	44.2	3.48 ± 0.02
	bp, matched	48.9	49.6	0.99 ± 0.00	22.5	28.0	0.80 ± 0.01
	bp, FDK weights	44.0	49.6	0.89 ± 0.01	17.6	28.0	0.63 ± 0.01
512^3	fp, Siddon	218	476	0.46 ± 0.00	176	361	0.49 ± 0.01
	fp, interpolated	1238	476	2.60 ± 0.02	1204	361	3.33 ± 0.02
	bp, matched	373	344	1.09 ± 0.01	172	216	0.79 ± 0.01
	bp, FDK weights	335	344	0.97 ± 0.01	130	216	0.60 ± 0.00
1024^3	fp, Siddon	1790	3754	0.48 ± 0.00	1469	3099	0.47 ± 0.01
	fp, interpolated	10080	3754	2.68 ± 0.02	9768	3099	3.14 ± 0.05
	bp, matched	3030	2664	1.14 ± 0.01	1426	1722	0.83 ± 0.00
	bp, FDK weights	2638	2664	0.99 ± 0.00	1024	1722	0.59 ± 0.00

Table 21: Linux, Quadro RTX 6000. Parallel beam (TIGRE parallel mode, ASTRA `parallel3d`), n^3, 360 angles, 1 GPU.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
256^3	fp, Siddon	51.3	54.5	0.95 ± 0.02	24.3	28.9	0.84 ± 0.01
	fp, interpolated	145	54.5	2.66 ± 0.01	136	28.9	4.72 ± 0.01
	bp	42.6	49.2	0.87 ± 0.00	15.2	26.6	0.57 ± 0.00
512^3	fp, Siddon	218	366	0.60 ± 0.00	173	252	0.68 ± 0.01
	fp, interpolated	1085	366	2.96 ± 0.03	1041	252	4.12 ± 0.02
	bp	319	326	0.98 ± 0.01	107	196	0.54 ± 0.00
1024^3	fp, Siddon	1739	2655	0.65 ± 0.00	1411	2004	0.70 ± 0.01
	fp, interpolated	8685	2655	3.26 ± 0.04	8361	2004	4.16 ± 0.06
	bp	2464	2512	0.98 ± 0.02	838	1566	0.54 ± 0.00

Table 22: Linux, Quadro RTX 6000. Fan beam (2D): TIGRE's one-slice cone geometry against ASTRA's `fanflat` (FP_CUDA, BP_CUDA, FBP_CUDA), n^2 image, n detector pixels, 1 GPU.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
512^2	fp, Siddon	2.50	1.11	2.39 ± 0.60	2.22	1.01	2.20 ± 0.00
	fp, interpolated	4.64	1.11	4.19 ± 0.04	4.36	1.01	4.31 ± 0.02
	bp, matched	2.48	3.33	0.74 ± 0.00	1.99	3.18	0.63 ± 0.00
	bp, FDK weights	2.45	3.33	0.73 ± 0.00	1.96	3.18	0.62 ± 0.00
1024^2	fp, Siddon	3.82	2.28	1.68 ± 0.01	3.11	1.95	1.60 ± 0.01
	fp, interpolated	11.6	2.28	5.10 ± 0.02	10.9	1.95	5.59 ± 0.01
	bp, matched	3.99	10.7	0.37 ± 0.00	3.16	10.3	0.31 ± 0.00
	bp, FDK weights	3.87	10.7	0.36 ± 0.00	3.03	10.3	0.29 ± 0.00
2048^2	fp, Siddon	9.44	6.40	1.48 ± 0.00	7.33	4.93	1.49 ± 0.00
	fp, interpolated	38.6	6.40	6.04 ± 0.05	36.5	4.93	7.42 ± 0.04
	bp, matched	10.3	40.2	0.26 ± 0.00	8.22	38.7	0.21 ± 0.00
	bp, FDK weights	9.67	40.2	0.24 ± 0.00	7.53	38.7	0.19 ± 0.00
4096^2	fp, Siddon	38.9	21.7	1.80 ± 0.02	32.2	17.3	1.86 ± 0.00
	fp, interpolated	151	21.7	6.98 ± 0.04	144	17.3	8.30 ± 0.01
	bp, matched	53.9	165	0.33 ± 0.00	28.9	155	0.19 ± 0.00
	bp, FDK weights	51.8	165	0.31 ± 0.00	26.2	155	0.17 ± 0.00
1024^2, 1440 angles	fp, Siddon	7.79	4.86	1.61 ± 0.00	7.16	4.56	1.57 ± 0.01
	fp, interpolated	38.1	4.86	7.85 ± 0.03	37.5	4.56	8.23 ± 0.00
	bp, matched	9.99	39.8	0.25 ± 0.00	8.55	39.1	0.22 ± 0.00
	bp, FDK weights	9.41	39.8	0.24 ± 0.00	7.92	39.1	0.20 ± 0.00

Table 23: Linux, Quadro RTX 6000. Cone beam 512^3, varying the number of angles, 1 GPU.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
45 angles	fp, Siddon	84.0	119	0.71 ± 0.01	35.8	58.9	0.61 ± 0.00
	fp, interpolated	212	119	1.78 ± 0.02	164	58.9	2.78 ± 0.01
	bp, matched	240	137	1.75 ± 0.00	28.4	30.9	0.92 ± 0.02
	bp, FDK weights	237	137	1.73 ± 0.01	19.3	30.9	0.62 ± 0.01
90 angles	fp, Siddon	98.5	165	0.59 ± 0.01	53.2	101	0.53 ± 0.00
	fp, interpolated	359	165	2.16 ± 0.04	314	101	3.12 ± 0.03
	bp, matched	259	163	1.59 ± 0.00	48.8	56.5	0.86 ± 0.01
	bp, FDK weights	248	163	1.53 ± 0.04	35.2	56.5	0.62 ± 0.00
180 angles	fp, Siddon	136	265	0.51 ± 0.01	94.1	187	0.50 ± 0.00
	fp, interpolated	655	265	2.46 ± 0.03	613	187	3.28 ± 0.02
	bp, matched	297	225	1.32 ± 0.00	90.5	110	0.83 ± 0.00
	bp, FDK weights	275	225	1.23 ± 0.01	66.7	110	0.61 ± 0.00
360 angles	fp, Siddon	220	477	0.46 ± 0.00	176	362	0.49 ± 0.01
	fp, interpolated	1251	477	2.62 ± 0.02	1209	362	3.34 ± 0.01
	bp, matched	380	346	1.10 ± 0.01	173	218	0.79 ± 0.01
	bp, FDK weights	342	346	$0.99 \pm 0.02^{\dagger}$	131	218	0.60 ± 0.01
720 angles	fp, Siddon	387	933	0.42 ± 0.01	343	732	0.47 ± 0.01
	fp, interpolated	2452	933	2.63 ± 0.01	2411	732	3.29 ± 0.02
	bp, matched	540	607	0.89 ± 0.00	337	430	0.78 ± 0.01
	bp, FDK weights	463	607	0.77 ± 0.02	258	430	0.60 ± 0.00
1440 angles	fp, Siddon	726	1822	0.40 ± 0.01	676	1495	0.45 ± 0.00
	fp, interpolated	4852	1822	2.66 ± 0.01	4813	1495	3.22 ± 0.01
	bp, matched	872	1075	0.81 ± 0.00	665	858	0.78 ± 0.00
	bp, FDK weights	721	1075	0.67 ± 0.00	514	858	0.60 ± 0.00

Table 24: Linux, Quadro RTX 6000. Non-cubic problems, 1 GPU: clinical CBCT ($256 \times 512 \times 512$ image, 512×1024 detector), fine detector (256^3 image, 1024^2 detector), coarse detector (512^3 image, 256^2 detector), 360 angles.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
clinical CBCT	fp, Siddon	332	813	0.41 ± 0.01	264	632	0.42 ± 0.00
	fp, interpolated	2325	813	2.86 ± 0.00	2279	632	3.60 ± 0.01
	bp, matched	214	243	0.88 ± 0.02	104	115	0.91 ± 0.01
	bp, FDK weights	184	243	0.75 ± 0.03	74.2	115	0.65 ± 0.00
fine detector	fp, Siddon	605	920	0.65 ± 0.01	276	601	0.46 ± 0.01
	fp, interpolated	2210	920	2.40 ± 0.02	2183	601	3.63 ± 0.02
	bp, matched	153	239	0.64 ± 0.01	47.1	52.6	0.90 ± 0.00
	bp, FDK weights	164	239	0.70 ± 0.05	40.6	52.6	0.77 ± 0.02
coarse detector	fp, Siddon	134	165	0.81 ± 0.01	90.3	101	0.90 ± 0.01
	fp, interpolated	426	165	2.58 ± 0.04	381	101	3.79 ± 0.00
	bp, matched	368	315	1.16 ± 0.01	160	209	0.77 ± 0.01
	bp, FDK weights	336	315	1.07 ± 0.01	125	209	0.60 ± 0.01

Table 25: Linux, Quadro RTX 6000. Cone beam 512^3 with the detector shifted sideways by 102.4 mm (a quarter of its width; ASTRA `cone_vec`), 1 GPU.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
u offset 102.4 mm	fp, Siddon	205	461	0.44 ± 0.01	145	347	0.42 ± 0.00
	fp, interpolated	1244	461	2.69 ± 0.02	1203	347	3.48 ± 0.02
	bp, matched	373	344	1.08 ± 0.00	168	216	0.78 ± 0.01
	bp, FDK weights	337	344	0.98 ± 0.01	129	216	0.60 ± 0.00

Table 26: Linux, Quadro RTX 6000. Calibration: the ASTRA convention that matches TIGRE, found on a small problem for each beam type, and how well the projections agree under it (relative L_2 difference after a least-squares scale). "Next best": the best convention that gives measurably different projections.

beam	ASTRA convention	agree	scale	next best
cone	$\theta_A = +1\,\theta_T + 90°$	5.4e-04	0.9998	1.6e-01
parallel	$\theta_A = +1\,\theta_T + 270°$, u flipped	5.1e-04	0.9998	4.9e-01
fan	$\theta_A = -1\,\theta_T + 90°$, u flipped	5.4e-04	0.9999	1.2e-01
offset	detector shift $+1\,u, +1\,v$	5.2e-04	0.9998	3.9e-01

Table 27: Linux, Quadro RTX 6000. The 3.3.0 note's detector (0.8 mm pixels whatever n), 1 GPU. Compare with Table 5 at the same n.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
256^3, 0.8 mm	fp, Siddon	51.2	66.6	0.77 ± 0.02	25.1	44.1	0.57 ± 0.00
	fp, interpolated	160	66.6	2.41 ± 0.00	154	44.1	3.49 ± 0.01
	bp, matched	47.9	50.1	0.96 ± 0.00	22.1	28.1	0.79 ± 0.00
	bp, FDK weights	44.5	50.1	0.89 ± 0.01	18.0	28.1	0.64 ± 0.01
1024^3, 0.8 mm	fp, Siddon	1679	3604	0.47 ± 0.00	1355	2950	0.46 ± 0.00
	fp, interpolated	11291	3604	3.12 ± 0.03	10990	2950	3.71 ± 0.05
	bp, matched	2877	2547	1.12 ± 0.01	1283	1667	0.77 ± 0.00
	bp, FDK weights	2621	2547	1.02 ± 0.01	1002	1667	0.60 ± 0.00

Table 28: Linux, Quadro RTX 6000. Small problems (cone beam, n^3 image, n^2 detector, 100 angles), 1 GPU.

		NumPy arrays, 1 GPU			PyTorch GPU tensors, 1 GPU		
problem	operation	TIGRE ms	ASTRA ms	ratio	TIGRE ms	ASTRA ms	ratio
32^3	fp, Siddon	2.20	3.17	0.69 ± 0.03	1.73	2.09	0.82 ± 0.06
	fp, interpolated	2.81	3.17	0.86 ± 0.06	2.31	2.09	1.10 ± 0.07
	bp, matched	2.16	2.40	0.90 ± 0.01	1.63	1.45	1.15 ± 0.07
	bp, FDK weights	2.15	2.40	0.89 ± 0.00	1.62	1.45	1.15 ± 0.11
64^3	fp, Siddon	2.26	3.75	0.59 ± 0.06	1.72	2.62	0.66 ± 0.00
	fp, interpolated	3.00	3.75	0.83 ± 0.12	2.64	2.62	1.01 ± 0.00
	bp, matched	2.13	2.37	0.90 ± 0.00	1.66	1.46	1.13 ± 0.00
	bp, FDK weights	2.12	2.37	0.90 ± 0.00	1.62	1.46	1.11 ± 0.00
128^3	fp, Siddon	3.71	6.91	0.54 ± 0.00	2.46	4.75	0.52 ± 0.00
	fp, interpolated	8.07	6.91	1.17 ± 0.00	7.10	4.75	1.50 ± 0.00
	bp, matched	3.64	4.18	0.87 ± 0.00	2.41	2.26	1.07 ± 0.00
	bp, FDK weights	3.55	4.18	0.85 ± 0.00	2.21	2.26	0.98 ± 0.00

Code availability

TIGRE: `https://github.com/CERN/TIGRE`, release 3.3.1. The comparison suite is the file `tigre_vs_astra_suite.py` in `docs/technical-notes/speed-3.3.1/`, whose `README.md` says how to run it on another machine. The raw results of this note and the scripts that make its tables are attached to the TIGRE 3.3.1 release on GitHub. Results from other GPUs and multi-GPU machines are welcome as an issue on TIGRE's GitHub.

Use of generative AI

This work and this note were done with substantial help from Claude (Anthropic), a large language model, used as a coding and writing assistant (Claude Opus, through Claude Code). Under the author's direction, Claude carried out much of the investigation and profiling, wrote the changes to TIGRE's host code and their tests, wrote the benchmark suite and the scripts that ran it, and wrote the first draft of this note.

The author conceived and directed the work, decided what to change and how, reviewed and merged every change, ran the benchmarks on GPU hardware (the results reported here), checked the results, and revised the text. The author takes full responsibility for the content of this note, including any errors. Claude is not an author: it cannot take responsibility for the work or for its publication.

References

[1] A. Biguri, M. Dosanjh, S. Hancock, M. Soleimani, "TIGRE: a MATLAB-GPU toolbox for CBCT image reconstruction," *Biomedical Physics & Engineering Express* 2(5):055010, 2016. doi:10.1088/2057-1976/2/5/055010.

[2] A. Biguri, R. J. Lindroos, R. Bryll, H. Towsyfyan, H. Deyhle, I. Harrane, R. Boardman, M. Mavrogordato, M. Dosanjh, S. Hancock, T. Blumensath, "Arbitrarily large tomography with iterative algorithms on multiple GPUs using the TIGRE toolbox," *Journal of Parallel and Distributed Computing* 146:52–63, 2020. doi:10.1016/j.jpdc.2020.07.004.

[3] A. Biguri, T. Sadakane, R. Lindroos, Y. Liu, M. Sabaté Landman, Y. Du, M. Lohvithee, S. Kaser, S. Hatamikia, R. Bryll, E. Valat, S. Wonglee, T. Blumensath, C.-B. Schönlieb,

"TIGRE v3: efficient and easy to use iterative computed tomographic reconstruction toolbox for real datasets," *Engineering Research Express* 7(1):015011, 2025. doi:10.1088/2631-8695/adbb3a.

[4] W. van Aarle, W. J. Palenstijn, J. Cant, E. Janssens, F. Bleichrodt, A. Dabravolski, J. De Beenhouwer, K. J. Batenburg, J. Sijbers, "Fast and flexible X-ray tomography using the ASTRA toolbox," *Optics Express* 24(22):25129–25147, 2016.

[5] A. Biguri, "Siddon–Jacobs ray tracing on GPUs: a technical note on a consistent and accurate implementation," TIGRE technical note, November 2026, revised for TIGRE 3.3.1.