Intel Xeon Platinum 8358 @ 2.60GHz + NVIDIA A100 80GB PCIe

Machine `xeon-platinum-8358`. The tables with a cost column measured on this machine, and its device tables.

10 table(s) in this directory. Each is shown under the file that produces it, at the manuscript's column width, so a table that does not fit is visible here.

`tab-gpu-call-h_nu_s.tex`

Table 1: GPU call execution cost for the (h, ν, s) datasets. Entries give the call cost in nanoseconds per element, host-device transfers included, for each implementation and dataset at its fastest data-transfer strategy; the final rows give the speedup of libmatern-cuda relative to the fastest external implementation for each dataset and its range from the quartiles of the repetitions. Each external implementation is shown in its best assembly for each dataset, the one with the lowest maximum ULP error among those with the fewest non-finite results, direct on an exact tie. Each cost is the median of the 25 timed repetitions of the call. *IQR*: the largest interquartile range of these repetitions over the datasets of the row, in per cent of the median. Results were obtained on the A100 GPU.

kernel	Grid	Random	Small z	Near-int ν	Half-int ν	Large z	Large ν	Seams	IQR
N (points timed)	262 144	250 000	100 000	100 200	100 000	100 000	150 000	100 000	
fastest transfer strategy	async-mapped	async-mapped	sync	sync	sync	sync	async-mapped	sync	
libmatern-cuda	4.31	5.17	7.12	7.15	6.80	7.24	6.92	7.18	1.0 %
[1] boost-gpu	4.34	5.47	7.35	8.03	7.32	8.01	18.4	7.35	1.5 %
[2] boost-gpu-modified	4.33	5.58	7.40	7.80	7.26	7.92	19.6	7.35	0.81 %
[3] kaust-128	10.9	11.9	15.4	14.7	14.9	14.6	16.6	14.7	0.76 %
[4] kaust-40	5.80	6.64	9.96	9.33	9.38	9.17	11.3	9.29	0.78 %
[5] kaust-modified	14.6	22.9	23.8	20.2	25.5	25.2	28.6	19.0	0.48 %
fastest external	[2]	[1]	[1]	[2]	[2]	[2]	[4]	[1]	
libmatern-cuda speed-up vs fastest external	1.00×	1.06×	1.03×	1.09×	1.07×	1.09×	1.62×	1.02×	
within-run bounds, low	1.0003×	1.05×	1.03×	1.08×	1.06×	1.09×	1.61×	1.02×	
within-run bounds, high	1.01×	1.07×	1.04×	1.10×	1.08×	1.11×	1.63×	1.03×	

fastest transfer strategy: the strategy of libmatern-cuda. [n]: key of an external implementation; the fastest-external row names the fastest external implementation of the dataset by its key. *within-run bounds*: the speed-up from the quartiles of the repetitions of one run, low: lower quartile of the external over upper quartile of the library, high: upper over lower; they describe the repetitions, not the variation between runs, which can be larger. `kaust-modified`: an adjusted configuration of the KAUST code, more accurate than the code as released (the adaptive integration range of its paper in place of $t \in [0, 9]$, 128 panels, the decade factor of the small-x recurrence applied, $\log\cosh$ without the intermediate cosh).

tab-gpu-call-z_nu.tex

Table 2: GPU call execution cost for the (z, ν) datasets. Entries give the call cost in nanoseconds per element, host-device transfers included, for each implementation and dataset at its fastest data-transfer strategy; the final rows give the speedup of libmatern-cuda relative to the fastest external implementation for each dataset and its range from the quartiles of the repetitions. Each external implementation is shown in its best assembly for each dataset, the one with the lowest maximum ULP error among those with the fewest non-finite results, direct on an exact tie. Each cost is the median of the 25 timed repetitions of the call. *IQR*: the largest interquartile range of these repetitions over the datasets of the row, in per cent of the median. Results were obtained on the A100 GPU.

kernel	Grid	Random	Small z	Near-int ν	Half-int ν	Large z	Large ν	Seams	IQR
N (points timed)	262 144	250 000	100 000	100 200	100 000	100 000	150 000	100 000	
fastest transfer strategy	async-mapped	async-mapped	sync	sync	sync	sync	async-mapped	sync	
libmatern-cuda	3.91	4.02	5.88	6.18	5.59	6.31	5.38	6.67	1.2 %
[1] boost-gpu	3.95	4.51	6.21	7.14	6.14	7.11	16.9	6.90	1.1 %
[2] boost-gpu-modified	3.91	4.63	6.24	6.90	6.10	7.07	18.1	6.90	0.88 %
[3] kaust-128	10.7	11.0	14.1	13.8	13.7	13.7	15.1	14.2	0.95 %
[4] kaust-40	5.62	5.72	8.77	8.36	8.20	8.30	9.75	8.83	0.98 %
[5] kaust-modified	14.5	22.0	22.7	19.3	24.4	24.4	27.2	18.5	0.83 %
fastest external	[2]	[1]	[1]	[2]	[2]	[2]	[4]	[1]	
libmatern-cuda speed-up vs fastest external	≈1.000×	1.12×	1.06×	1.12×	1.09×	1.12×	1.81×	1.03×	
within-run bounds, low	0.991×	1.12×	1.04×	1.10×	1.08×	1.11×	1.80×	1.03×	
within-run bounds, high	1.01×	1.13×	1.07×	1.12×	1.10×	1.13×	1.83×	1.05×	

fastest transfer strategy: the strategy of libmatern-cuda. [n]: key of an external implementation; the fastest-external row names the fastest external implementation of the dataset by its key. *within-run bounds*: the speed-up from the quartiles of the repetitions of one run, low: lower quartile of the external over upper quartile of the library, high: upper over lower; they describe the repetitions, not the variation between runs, which can be larger. ≈: the interquartile ranges of the library and of the fastest external implementation overlap, so within one run the repetitions do not resolve the speed-up. `kaust-modified`: an adjusted configuration of the KAUST code, more accurate than the code as released (the adaptive integration range of its paper in place of $t \in [0, 9]$, 128 panels, the decade factor of the small-x recurrence applied, log cosh without the intermediate cosh).

`tab-gpu-cost-h_nu_s.tex`

Table 3: GPU kernel execution cost for the (h, ν, s) datasets. Entries give the kernel cost in nanoseconds per element for each implementation and dataset; the final rows give the speedup of libmatern-cuda relative to the fastest external implementation for each dataset and its range from the quartiles of the repetitions. Each external implementation is shown in its best assembly for each dataset, the one with the lowest maximum ULP error among those with the fewest non-finite results, direct on an exact tie. Each cost is the median of the 25 timed repetitions of the kernel. *IQR*: the largest interquartile range of these repetitions over the datasets of the row, in per cent of the median. Results were obtained on the A100 GPU.

kernel	Grid	Random	Small z	Near-int ν	Half-int ν	Large z	Large ν	Seams	IQR
N (points timed)	262 144	250 000	100 000	100 200	100 000	100 000	150 000	100 000	
libmatern-cuda	0.285	0.795	0.461	0.542	0.174	0.614	0.819	0.492	2.5 %
boost-gpu	0.457	1.36	0.922	1.57	0.860	1.59	12.5	0.870	4.5 %
boost-gpu-modified	0.605	1.54	0.973	1.35	0.850	1.51	13.8	0.891	2.1 %
kaust-128	7.54	7.95	8.84	8.22	8.42	8.17	10.8	8.17	0.95 %
kaust-40	2.48	2.65	3.47	2.83	2.94	2.70	5.47	2.78	2.5 %
kaust-modified	11.3	18.8	17.4	13.7	19.0	18.8	22.8	12.5	0.97 %
libmatern-cuda speed-up vs fastest external	1.60×	1.71×	2.00×	2.49×	4.88×	2.45×	6.68×	1.77×	
within-run bounds, low	1.58×	1.68×	1.98×	2.47×	4.82×	2.43×	6.52×	1.75×	
within-run bounds, high	1.62×	1.72×	2.05×	2.56×	4.88×	2.51×	6.86×	1.81×	

within-run bounds: the speed-up from the quartiles of the repetitions of one run, low: lower quartile of the external over upper quartile of the library, high: upper over lower; they describe the repetitions, not the variation between runs, which can be larger. `kaust-modified`: an adjusted configuration of the KAUST code, more accurate than the code as released (the adaptive integration range of its paper in place of $t \in [0, 9]$, 128 panels, the decade factor of the small-x recurrence applied, $\log\cosh$ without the intermediate cosh).

`tab-gpu-cost-z_nu.tex`

Table 4: GPU kernel execution cost for the (z, ν) datasets. Entries give the kernel cost in nanoseconds per element for each implementation and dataset; the final rows give the speedup of libmatern-cuda relative to the fastest external implementation for each dataset and its range from the quartiles of the repetitions. Each external implementation is shown in its best assembly for each dataset, the one with the lowest maximum ULP error among those with the fewest non-finite results, direct on an exact tie. Each cost is the median of the 25 timed repetitions of the kernel. *IQR*: the largest interquartile range of these repetitions over the datasets of the row, in per cent of the median. Results were obtained on the A100 GPU.

kernel	Grid	Random	Small z	Near-int ν	Half-int ν	Large z	Large ν	Seams	IQR
N (points timed)	262 144	250 000	100 000	100 200	100 000	100 000	150 000	100 000	
libmatern-cuda	0.273	0.778	0.440	0.531	0.164	0.594	0.799	0.471	2.3 %
boost-gpu	0.453	1.35	0.911	1.56	0.860	1.58	12.5	0.860	4.6 %
boost-gpu-modified	0.602	1.52	0.963	1.35	0.840	1.50	13.7	0.881	1.4 %
kaust-128	7.54	7.95	8.83	8.22	8.41	8.16	10.8	8.17	1.1 %
kaust-40	2.47	2.65	3.47	2.84	2.93	2.69	5.43	2.76	1.5 %
kaust-modified	11.3	18.8	17.4	13.8	19.1	18.8	22.8	12.5	1.2 %
libmatern-cuda speed-up vs fastest external	1.66×	1.73×	2.07×	2.54×	5.12×	2.52×	6.79×	1.83×	
within-run bounds, low	1.62×	1.70×	2.02×	2.52×	5.06×	2.46×	6.69×	1.79×	
within-run bounds, high	1.66×	1.75×	2.09×	2.54×	5.12×	2.53×	6.91×	1.83×	

within-run bounds: the speed-up from the quartiles of the repetitions of one run, low: lower quartile of the external over upper quartile of the library, high: upper over lower; they describe the repetitions, not the variation between runs, which can be larger. `kaust-modified`: an adjusted configuration of the KAUST code, more accurate than the code as released (the adaptive integration range of its paper in place of $t \in [0, 9]$, 128 panels, the decade factor of the small-x recurrence applied, $\log\cosh$ without the intermediate cosh).

`tab-gpu-exclusions-h_nu_s.tex`

Table 5: Points at which a GPU kernel returned no finite number, for the (h, ν, s) datasets. Results were obtained on the A100 GPU.

kernel	Grid	Random	Small z	Near-int ν	Half-int ν	Large z	Large ν	Seams
N (points)	262 144	250 000	100 000	100 200	100 000	100 000	150 000	100 000
libmatern-cuda	0	0	0	0	0	0	0	0
boost-gpu	0	0	11 856	0	0	0	47 474	0
boost-gpu-modified	0	0	11 856	0	0	0	47 474	0
kaust-128	0	0	0	0	0	21 076	86 860	0
kaust-40	0	0	0	0	0	21 076	86 860	0
kaust-modified	0	0	0	0	0	0	0	0

`kaust-modified`: an adjusted configuration of the KAUST code, more accurate than the code as released (the adaptive integration range of its paper in place of $t \in [0, 9]$, 128 panels, the decade factor of the small-x recurrence applied, $\log\cosh$ without the intermediate cosh).

`tab-gpu-exclusions-z_nu.tex`

Table 6: Points at which a GPU kernel returned no finite number, for the (z, ν) datasets. Results were obtained on the A100 GPU.

kernel	Grid	Random	Small z	Near-int ν	Half-int ν	Large z	Large ν	Seams
N (points)	262 144	250 000	100 000	100 200	100 000	100 000	150 000	100 000
libmatern-cuda	0	0	0	0	0	0	0	0
boost-gpu	0	0	11 856	0	0	0	47 474	0
boost-gpu-modified	0	0	11 856	0	0	0	47 474	0
kaust-128	0	0	0	0	0	21 076	86 860	0
kaust-40	0	0	0	0	0	21 076	86 860	0
kaust-modified	0	0	0	0	0	0	0	0

`kaust-modified`: an adjusted configuration of the KAUST code, more accurate than the code as released (the adaptive integration range of its paper in place of $t \in [0, 9]$, 128 panels, the decade factor of the small-x recurrence applied, $\log\cosh$ without the intermediate cosh).

tab-gpu-medmax-h_nu_s.tex

Table 7: Median and maximum ULP error (median / maximum) over all scored points of the GPU kernels for the (h, ν, s) datasets. Each external implementation is shown in its best assembly for each dataset, the one with the lowest maximum ULP error among those with the fewest non-finite results, direct on an exact tie. Results were obtained on the A100 GPU.

kernel	Grid	Random	Small z	Near-int ν	Half-int ν	Large z	Large ν	Seams	finite
libmatern-cuda	1.24 / 13.6	1.05 / 24.1	0.483 / 7.24	0.865 / 16.2	0.597 / 8.06	0.751 / 21.1	0.731 / 26.8	1.07 / 65.7	8/8
boost-gpu	24.7 / 4.48×10^5	20.1 / *Cap*	2.00 / 21.8†	2.15 / 53.4	2.57 / *Cap*	79.1 / *Cap*	150 / *Cap*†	2.55 / 1469	3/8
boost-gpu-modified	12.8 / 295	3.17 / 82.2	2.00 / 21.8†	2.15 / 53.4	2.57 / *Cap*	79.1 / *Cap*	150 / *Cap*†	2.55 / 1431	4/8
kaust-128	29.8 / 3939	8.14 / *Cap*	5.84 / *Cap*	4.76 / *Cap*	7.89 / *Cap*	2.75×10^{11} / *Cap*†	59.6 / *Cap*†	6.39 / *Cap*	1/8
kaust-40	2.15×10^{13} / *Cap*	4.26×10^{10} / *Cap*	6.27 / *Cap*	1.05×10^4 / *Cap*	2.06×10^8 / *Cap*	2.35×10^{15} / *Cap*†	1.04×10^{13} / *Cap*†	4.63×10^8 / *Cap*	0/8
kaust-modified	21.3 / 379	8.97 / 277	59.8 / 2070	4.75 / 321	43.1 / 1686	179 / 2568	332 / 3.19×10^4	5.80 / 1996	8/8
Lowest external	boost-gpu-mod.	boost-gpu-mod.	boost-gpu/ kaust-mod.	boost-gpu	boost-gpu/ kaust-mod.	boost-gpu-mod./ kaust-mod.	kaust-128/ kaust-mod.	boost-gpu-mod.	–
Advantage factor	lib 10.4/21.6	lib 3.02/3.42	lib 4.14/286	lib 2.49/3.30	lib 4.31/209	lib 105/122	lib 81.5/1191	lib 2.38/21.8	–

Cap: a maximum at or above the 10^6 ULP cap; the value is in the `.csv`. †: maximum over a reduced population, non-finite results excluded. *finite*: number of datasets with a finite result at every point and a maximum below 10^6 ULP. *Lowest external*: the external implementation with the lowest median and the one with the lowest maximum (median/maximum; one name where both coincide). The median is ranked over all external implementations, rows marked † or *Cap* included; the maximum only over those with a finite result at every point and a maximum below the 10^6 ULP cap. *Advantage factor*: for the median and the maximum, the ratio of the larger to the smaller of the lowest external value and this library's, labelled with the implementation it favours (`lib`: this library). `kaust-modified`: an adjusted configuration of the KAUST code, more accurate than the code as released (the adaptive integration range of its paper in place of $t \in [0, 9]$, 128 panels, the decade factor of the small-x recurrence applied, $\log\cosh$ without the intermediate cosh).

tab-gpu-medmax-z_nu.tex

Table 8: Median and maximum ULP error (median / maximum) over all scored points of the GPU kernels for the (z, ν) datasets. Each external implementation is shown in its best assembly for each dataset, the one with the lowest maximum ULP error among those with the fewest non-finite results, direct on an exact tie. Results were obtained on the A100 GPU.

kernel	Grid	Random	Small z	Near-int ν	Half-int ν	Large z	Large ν	Seams	finite
libmatern-cuda	1.22 / 13.5	1.04 / 23.9	0.483 / 7.13	0.861 / 16.7	0.567 / 6.96	0.740 / 21.3	0.729 / 25.1	1.06 / 65.4	8/8
boost-gpu	20.7 / 4.48×10^5	19.2 / *Cap*	2.00 / 21.8†	1.96 / 24.4	1.52 / *Cap*	1.55 / *Cap*	148 / *Cap*†	1.66 / 1371	3/8
boost-gpu-modified	1.29 / 21.6	1.67 / 29.7	2.00 / 21.8†	1.96 / 24.4	1.52 / *Cap*	1.55 / *Cap*	148 / *Cap*†	1.66 / 101	4/8
kaust-128	26.0 / 3897	7.43 / *Cap*	5.83 / *Cap*	4.70 / *Cap*	7.61 / *Cap*	2.75×10^{11} / *Cap*†	58.2 / *Cap*†	6.03 / *Cap*	1/8
kaust-40	2.15×10^{13} / *Cap*	4.26×10^{10} / *Cap*	6.25 / *Cap*	1.05×10^4 / *Cap*	2.06×10^8 / *Cap*	2.35×10^{15} / *Cap*†	1.04×10^{13} / *Cap*†	4.63×10^8 / *Cap*	0/8
kaust-modified	17.2 / 260	8.47 / 277	59.8 / 2070	4.72 / 321	42.1 / 1327	158 / 1812	329 / 3.19×10^4	5.55 / 994	8/8
Lowest external	boost-gpu-mod.	boost-gpu-mod.	boost-gpu/ kaust-mod.	boost-gpu	boost-gpu/ kaust-mod.	boost-gpu-mod./ kaust-mod.	kaust-128/ kaust-mod.	boost-gpu-mod.	–
Advantage factor	lib 1.06/1.60	lib 1.61/1.24	lib 4.14/290	lib 2.28/1.46	lib 2.67/191	lib 2.10/85.1	lib 79.9/1268	lib 1.57/1.54	–

Cap: a maximum at or above the 10^6 ULP cap; the value is in the `.csv`. †: maximum over a reduced population, non-finite results excluded. *finite*: number of datasets with a finite result at every point and a maximum below 10^6 ULP. *Lowest external*: the external implementation with the lowest median and the one with the lowest maximum (median/maximum; one name where both coincide). The median is ranked over all external implementations, rows marked † or *Cap* included; the maximum only over those with a finite result at every point and a maximum below the 10^6 ULP cap. *Advantage factor*: for the median and the maximum, the ratio of the larger to the smaller of the lowest external value and this library's, labelled with the implementation it favours (`lib`: this library). `kaust-modified`: an adjusted configuration of the KAUST code, more accurate than the code as released (the adaptive integration range of its paper in place of $t \in [0, 9]$, 128 panels, the decade factor of the small-x recurrence applied, $\log\cosh$ without the intermediate cosh).

`tab-gpu-summary-h_nu_s.tex`

Table 9: GPU cost and speed-up of libmatern-cuda for the (h, ν, s) datasets. Kernel columns give the device time of the kernel alone, end-to-end columns the call with host-device staging and transfers, at each implementation's fastest transfer strategy; both in nanoseconds per element. The speed-up is over the fastest external implementation of the dataset and timing mode, not necessarily the same one in both; low and high are its within-run quartile bounds. Each external implementation is shown in its best assembly for each dataset, the one with the lowest maximum ULP error among those with the fewest non-finite results, direct on an exact tie. Each cost is the median of the 25 timed repetitions of the kernel launch or call. Results were obtained on the A100 GPU.

	kernel				end-to-end call				
dataset	ns/value	speed-up	low	high	ns/value	speed-up	low	high	IQR
Grid	0.285	1.60×	1.58×	1.62×	4.31	1.00×	1.0003×	1.01×	2.6 %
Random	0.795	1.71×	1.68×	1.72×	5.17	1.06×	1.05×	1.07×	1.5 %
Small z	0.461	2.00×	1.98×	2.05×	7.12	1.03×	1.03×	1.04×	2.2 %
Near-integer ν	0.542	2.49×	2.47×	2.56×	7.15	1.09×	1.08×	1.10×	1.9 %
Half-integer ν	0.174	4.88×	4.82×	4.88×	6.80	1.07×	1.06×	1.08×	1.2 %
Large z	0.614	2.45×	2.43×	2.51×	7.24	1.09×	1.09×	1.11×	1.7 %
Large ν	0.819	6.68×	6.52×	6.86×	6.92	1.62×	1.61×	1.63×	2.5 %
Router seams	0.492	1.77×	1.75×	1.81×	7.18	1.02×	1.02×	1.03×	2.1 %

Fastest external: `boost-gpu` (call: Random, Small z, Router seams; kernel: Grid, Random, Small z, Router seams); `boost-gpu-modified` (call: Grid, Near-integer ν, Half-integer ν, Large z; kernel: Near-integer ν, Half-integer ν, Large z); `kaust-40` (call: Large ν; kernel: Large ν). IQR: the largest interquartile range of the row's four timed cells (library and fastest external, both modes), in per cent of the median. *within-run bounds*: the speed-up from the quartiles of the repetitions of one run, low: lower quartile of the external over upper quartile of the library, high: upper over lower; they describe the repetitions, not the variation between runs, which can be larger.

`tab-gpu-summary-z_nu.tex`

Table 10: GPU cost and speed-up of libmatern-cuda for the (z, ν) datasets. Kernel columns give the device time of the kernel alone, end-to-end columns the call with host-device staging and transfers, at each implementation's fastest transfer strategy; both in nanoseconds per element. The speed-up is over the fastest external implementation of the dataset and timing mode, not necessarily the same one in both; low and high are its within-run quartile bounds. Each external implementation is shown in its best assembly for each dataset, the one with the lowest maximum ULP error among those with the fewest non-finite results, direct on an exact tie. Each cost is the median of the 25 timed repetitions of the kernel launch or call. Results were obtained on the A100 GPU.

	kernel				end-to-end call				
dataset	ns/value	speed-up	low	high	ns/value	speed-up	low	high	IQR
Grid	0.273	1.66×	1.62×	1.66×	3.91	≈1.000×	0.991×	1.01×	1.4 %
Random	0.778	1.73×	1.70×	1.75×	4.02	1.12×	1.12×	1.13×	1.6 %
Small z	0.440	2.07×	2.02×	2.09×	5.88	1.06×	1.04×	1.07×	2.3 %
Near-integer ν	0.531	2.54×	2.52×	2.54×	6.18	1.12×	1.10×	1.12×	0.95 %
Half-integer ν	0.164	5.12×	5.06×	5.12×	5.59	1.09×	1.08×	1.10×	1.2 %
Large z	0.594	2.52×	2.46×	2.53×	6.31	1.12×	1.11×	1.13×	1.7 %
Large ν	0.799	6.79×	6.69×	6.91×	5.38	1.81×	1.80×	1.83×	1.7 %
Router seams	0.471	1.83×	1.79×	1.83×	6.67	1.03×	1.03×	1.05×	2.2 %

Fastest external: `boost-gpu` (call: Random, Small z, Router seams; kernel: Grid, Random, Small z, Router seams); `boost-gpu-modified` (call: Grid, Near-integer ν, Half-integer ν, Large z; kernel: Near-integer ν, Half-integer ν, Large z); `kaust-40` (call: Large ν; kernel: Large ν). IQR: the largest interquartile range of the row's four timed cells (library and fastest external, both modes), in per cent of the median. *within-run bounds*: the speed-up from the quartiles of the repetitions of one run, low: lower quartile of the external over upper quartile of the library, high: upper over lower; they describe the repetitions, not the variation between runs, which can be larger. ≈: the interquartile ranges of the library and of the fastest external implementation overlap, so within one run the repetitions do not resolve the speed-up.