Performance Tuning of Deep Learning Framework Chainer on the K computer

Akiyoshi KURODA¹, Kiyoshi KUMAHATA¹, Syuichi CHIBA², Katsutoshi TAKASHINA² and Kazuo MINAMI¹ ¹ R-CCS, Operations and Computer Technologies Division, Application Tuning Development Unit. ² FUJITSU LIMITED.



1. Introduction

Recently GPUs has become a popular p	latform for executing deep learning	(DL) workloads. We	e revisit the idea of doing DL on CPUs ,
especially massively parallel CPU cluste	rs (supercomputers). In anticipatio	of deployment of t	he Supercomputer Fugaku with much
more DL capable CPUs, we investigate	which optimizations can be already	done using the K co	omputer, current leadership
computing facility and predecessor t			
2. Characterist			1 Profiler (cProfile) result on the K computer.
– Profiling res	ults of Chainer using cProfile+gpro	2do : • Original Chainer Ver.4.4.0	profile for MNIST sample (unit=1,000, epoch=20) on the K computer
are [Fig.1][ncalls tottime per	call cumtime percall filename:lineno(function)
train yoursel 12 main	tion time of 10.311 s breaks down as for	<u> </u>	.117 8,421.99 0.117 optimizers/adam.py:91(update_core_cpu)
– Adam op	timizer [adam.py]: 84%1.		.011 1,122.61 0.011 {method 'dot' of 'numpy.ndarray' objects} .002 47.38 0.002 activation/relu.py:29(forward cpu)
9937%	ot called from linear.py: 11%2.		.002 47.38 0.002 activation/relu.py:29(forward_cpu) .000 1,421.38 0.007 function_node.py:201(apply)
			.001 32.05 0.001 activation/relu.py:96(forward_cpu)
3. Adam opti			
Optimizer consumed 84% of execution time and ran with 0.04% efficiency as measured using Fujitsu			
hardware counters.			
— — — — — The domina	ant operation is square root of ma	rix elements called	d from NumPy for filter update.
- This functio	on in NumPy was implemented with	automatic C langu	age code generation, and thus
difficult to tune directlý			
📑 📻 🐂 🖯 — We rewrote all calculations in Adam using vectorized Fortran library and SIMD conversion and software			
pipelining (SWPL).			
📁 👘 – İn filter update calculations many values happened to be close to zero(denormalized number), raising			
underflow exceptions. We recompiled Python with an option forcing to truncate such numbers to			
🕂 🚔 🖉 👘 🖉 📮 We have also applied SWPL and masked SIMD using Fortran Library to implement ReLU activation			
Line# Hits Time Per Hit % Time Line Contents			
	t GEMM convolution		
- achieves only 7.76% peak performance. 104 7200 4482481.0 622.6 21.2 m += (1 - hp.beta1)* (grad - m) 105 7200 4366618.0 606.5 20.6 v += (1 - hp.beta2)* (grad * grad - v)			
- NumPy was compiled against vectorized (using 7200 7337882.0 1019.2 34.6 param.data = hp.eta * (self.l.* m / (numpy.sqrt(vhat) + hp.eps) +			
	single-threaded Fujitsu BLAS	113 7200 4864633.0 675.6	5 23.0 hp.weight_decay_rate * param.data)
			Fig.2 Line cost of Adam optimizer.
	d numpy.dot to call multi-threaded		
Fig.1 Call graph sample of Chainer		ImPy.dot calculation.	· · · · · · · · · · · · · · · · · · ·
on the K computer	subroutine calculation	emm size(M,N,K)	#call elapse [s] efficency %
Table.3 Elapsed time of tuning stage.	linear.py:33(forward) y=x.dot(W.T)	100, 1000, 784), (100, 1000,	
name tuning GEMM Adam other total	linear.py:96(forward) gx=gy.dot(W)	100, 1000, 10), (100, 1000,	
original 1,189.1 8,657.4 465.2 10,311.7	linear.py:145(forward) gW=gy.T.dot(x)	10, 1000, 100) , (1000, 1000	, 100), (1000, 784, 100) x36000 380.6[s] 8.90%
underflow Kfast(Kns) 990.2 1,681.9 372.2 3,044.4	Table.4 Performance efficiency of GEMM and s	aut .	Table.5 Profiler result after all tuning.
		/r L.	5
GEMM thread 236.5 8,386.9 336.9 8,960.2 Adam FortranLib. 1,052.6 780.7 439.6 2,272.9	name tuning GEMM sqrt	· .	rofile for MNIST sample (unit=1,000, epoch=20) on the K computer
	original 7.76% 0.04%	· .	
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2		Tuned Chainer Ver.4.4.0 p ncalls tottime perc 102000 166.80 0	cumtime percall filename:lineno(function) .002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects}
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result	original 7.76% 0.04% underflow Kfast(Kns) 9.31% 0.19% GEMM thread 38.81% 0.48% Adam FortranLb 8.76% 0.92%	• Tuned Chainer Ver.4.4.0 pr ncalls tottime perc 102000 166.80 0 72000 60.73 0	call cumtime percall filename:lineno(function) .002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} .001 104.47 0.001 optimizers/adam.py:53(adam_kro2)
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3]	original 7.76% 0.04% underflow Kfast(Kns) 9.31% 0.19% GEMM thread 38.81% 0.48% Adam FortranLib. 8.76% 0.92% all 47.03% 17.89%	Tuned Chainer Ver.4.4.0 p ncalls tottime perc 102000 166.80 0 72000 60.73 0 218000 38.49 0	call cumtime percall filename:lineno(function) 0.002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} 0.001 104.47 0.001 optimizers/adam.py:53(adam_kro2) 0.000 440.19 0.002 {method 'not on pode.py:201(apply)
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu	original 7.76% 0.04% underflow Kfast(Kns) 9.31% 0.19% GEMM thread 38.81% 0.48% Adam FortranLib. 8.76% 0.92% all 47.03% 17.89%	• Tuned Chainer Ver.4.4.0 p ncalls tottime per 102000 166.80 0 72000 60.73 0 218000 38.49 0 12000 18.17 0	call cumtime percall filename:lineno(function) .002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} .001 104.47 0.001 optimizers/adam.py:53(adam_kro2)
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ad	3]: underflow Kfast(Kns) 9.31% 0.49% GEMM thread 33.81% 0.48% Adam FortranLib. 8.76% 0.92% all 47.03% 17.89% Impy.dot: 1.15x dam's sqrt: 4.54x	• Tuned Chainer Ver.4.4.0 p ncalls tottime perc 102000 166.80 0 72000 60.73 0 218000 38.49 0 12000 18.17 0 364000 16.73 0	cumtime percall filename:lineno(function) .002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} .001 104.47 0.001 optimizers/adam.py:53(adam_kro2) .000 440.19 0.002 {inction_node.py:201(apply) .002 292.06 0.024 variable.py:968(_backward_main) .000 21.16 0.000 numpy/ctypeslib.py:196(from_param)
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3] • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control:	3]: underflow Kfast(Kns) 9.31% 0.49% GEMM thread 33.81% 0.48% Adam FortranLib. 8.76% 0.92% all 47.03% 17.89% Impy.dot: 1.15x dam's sqrt: 4.54x	• Tuned Chainer Ver.4.4.0 p ncalls tottime perc 102000 166.80 0 72000 60.73 0 218000 38.49 0 12000 18.17 0 364000 16.73 0	call cumtime percall filename:lineno(function) 0.002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} 0.001 104.47 0.001 optimizers/adam.py:53(adam_kro2) 0.000 440.19 0.002 function_node.py:201(apply) 0.002 292.06 0.024 variable.py:968(_backward_main)
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38> • Total speedup: 16.2x	3]: underflow Kfast(Kns) 9.31% 0.49% GEMM thread 33.81% 0.48% Adam FortranLib. 8.76% 0.92% all 47.03% 17.89% Impy.dot: 1.15x dam's sqrt: 4.54x	• Tuned Chainer Ver.4.4.0 p ncalls tottime perc 102000 166.80 0 72000 60.73 0 218000 38.49 0 12000 18.17 0 364000 16.73 0 6. Paralleliz We also	call cumtime percall filename:lineno(function) .002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} .001 104.47 0.001 optimizers/adam.py:53(adam_kro2) .000 440.19 0.002 function_node.py:201(apply) .002 292.06 0.024 variable.py:968(_backward_main) .000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN tried to install ChainerMN-1.3.0 and
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3] • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]:	3]: Impy.dot: 1.15x dam's sqrt: 4.54x	• Tuned Chainer Ver.4.4.0 p ncalls tottime perc 102000 166.80 0 72000 60.73 0 218000 38.49 0 12000 16.73 0 364000 16.73 0 G. Paralleliz We also released	all cumtime percall filename:lineno(function) .002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} .001 104.47 0.001 optimizers/adam.py:53(adam_kro2) .000 440.19 0.002 function_node.py:201(apply) .000 292.06 0.024 variable.py:968(_backward_main) .000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN tried to install ChainerMN-1.3.0 and tit to the K computer users. tit to the K computer users.
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3] • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x • Efficiency improvements [Table 4]: • • Using SSL II thread parallel BLAS in nu	3]: riginal 7.76% 0.04% underflow Kfast(Kns) 9.31% 0.19% GEMM thread 38.81% 0.48% Adam FortranLib. 8.76% 0.92% all 47.03% 17.89% the sqrt: 4.54x rightarrow 47.03%	• Tuned Chainer Ver.4.4.0 p ncalls tottime perc 102000 166.80 0 72000 60.73 0 218000 38.49 0 12000 18.17 0 364000 16.73 0 6. Paralleliz We also released 6.06x) – Scalabilit	call cumtime percall filename:lineno(function) .002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} .001 104.47 0.001 optimizers/adam.py:53(adam_kro2) .000 440.19 0.002 function_node.py:201(apply) .002 292.06 0.024 variable.py:968(_backward_main) .000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN tried to install ChainerMN-1.3.0 and
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3] • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]: Using SSL II thread parallel BLAS in nu • Using SSWPL by Fortran library for Ac	3]: Impy.dot: 1.15x dam's sqrt: 4.54x	• Tuned Chainer Ver.4.4.0 p ncalls tottime perc 102000 166.80 0 72000 60.73 0 218000 38.49 0 12000 18.17 0 364000 16.73 0 6. Paralleliz We also released 6.06x) - Scalabilit [Fig.4].	all cumtime percall filename:lineno(function) .002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} .001 104.47 0.001 optimizers/adam.py:53(adam_kro2) .000 440.19 0.002 {function_node.py:201(apply) .000 292.06 0.024 variable.py:968(_backward_main) .000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN tried to install ChainerMN-1.3.0 and dit to the K computer users. try of ChainerMN on the K computer is tried to install chainerMN on the K computer is
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3] • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x • Efficiency improvements [Table 4]: • • Using SSL II thread parallel BLAS in nu • Using SSL II thread parallel BLAS in Ac • Output 16.2x • Efficiency improvements [Table 4]: • Using of SWPL by Fortran library for Ac • 0.04%→0.92%→17.89% (490x)	3]: riginal 7.76% 0.04% underflow Kfast(Kns) 9.31% 0.19% GEMM thread 38.81% 0.48% Adam FortranLib. 8.76% 0.92% all 47.03% 17.89% all 47.03% 17.89% dam's sqrt: 4.54x c	• Tuned Chainer Ver.4.4.0 p ncalls tottime perc 102000 166.80 0 218000 38.49 0 12000 16.73 0 364000 16.73 0 6. Paralleliz We also released Scalabiliti control: [Fig.4]. • Measure	ail cumtime percall filename:lineno(function) .002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} .001 104.47 0.001 optimizers/adam.py:53(adam_kro2) .000 440.19 0.002 function_node.py:201(apply) .002 292.06 0.024 variable.py:968(_backward_main) .000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN 1.3.0 and dit to the K computer users. .oy of ChainerMN on the K computer is ment conditions: MNIST sample (unit=1000,
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3] • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x • Efficiency improvements [Table 4]: • • Using SSL II thread parallel BLAS in nu • Using SSL II thread parallel BLAS in nu • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • 0.04%→0.92%→17.89% (490x)	3]: riginal 7.76% 0.04% riginal 7.76% 0.04% riginal 7.76% 0.04% riginal 7.76% 0.04% riginal 7.76% 0.19% riginal 7.76% 0.19% riginal 7.76% 0.93% riginal 7.76% 0.93% riginal 7.76% 0.93% riginal 7.76% 0.92% riginal 7.76% 0.92%	 Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li>	allcumtimepercallfilename:lineno(function).002166.800.002 {method 'dot' of 'numpy.ndarray' objects}.001104.470.001 optimizers/adam.py:53(adam_kro2).000440.190.002 function_node.py:201(apply).002292.060.024 variable.py:968(_backward_main).00021.160.000 numpy/ctypeslib.py:196(from_param)zation by the ChainerMNtried to install ChainerMN-1.3.0 anddiet to the K computer users00computer users00computer users00MNIST sample (unit=1000, 20 → 1epoch=600iter).
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]: • • Using of SWPL by Fortran library for Ac • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • 0.04%→0.92%→17.89% (490x) Performance tuning of Chainer elapsed[s] • on the K computer	3]: $rac{riginal}{rac}$ 7.76% 0.04% underflow Kfast(Kns) 9.31% 0.19% GEMM thread 38.81% 0.48% Adam FortranLib. 8.76% 0.92% all 47.03% 17.89% table 47.03% 17.89% dam's sqrt: 4.54x c umpy.dot: 7.76% 38.81% 47.03% dam's sqrt and floating point underflow Scalability of Chainer on the K computer	 Tuned Chainer Ver.4.4.0 p ncalls tottime percentric percentri percentric percentric	ail cumtime percall filename:lineno(function) .002 166.80 0.002 (method 'dot' of 'numpy.ndarray' objects) .001 104.47 0.001 optimizers/adam.py:53(adam_kro2) .000 440.19 0.002 function_node.py:201(apply) .000 292.06 0.024 variable.py:968(_backward_main) .000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN tried to install ChainerMN-1.3.0 and d it to the K computer users. .y of ChainerMN on the K computer is ment conditions: MNIST sample (unit=1000, 20 → 1epoch=600iter). .0 1 can be measured even using 600 proc. or
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3] • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]: • • Using SSL II thread parallel BLAS in nu • Using SSL II thread parallel BLAS in nu • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • 0.04%→0.92%→17.89% (490x)	all $\frac{1}{2,76\%}$ 0.04% underflow Kfast(Kns) 9.31% 0.19% GEMM thread 38.81% 0.48% Adam FortranLib. 8.76% 0.92% all 47.03% 17.89% umpy.dot: 1.15x dam's sqrt: 4.54x C umpy.dot: 7.76%→38.81%→47.03% dam's sqrt and floating point underflow Scalability of Chainer	 Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li>	allcumtimepercallfilename:lineno(function).002166.800.002 {method 'dot' of 'numpy.ndarray' objects}.001104.470.001 optimizers/adam.py:53(adam_kro2).000440.190.002 function_node.py:201(apply).002292.060.024 variable.py:968(_backward_main).00021.160.000 numpy/ctypeslib.py:196(from_param)zation by the ChainerMNtried to install ChainerMN-1.3.0 anddiet to the K computer users00computer users00computer users00MNIST sample (unit=1000, 20 → 1epoch=600iter).
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]: • • Using of SWPL by Fortran library for Ac • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • O.04%→0.92%→17.89% (490x) Performance tuning of Chainer etapsed[s] on the K computer	3]: all $\frac{1,76\%}{4,38,1\%}$ 0.04% <u>GEMM</u> thread 38.81% 0.48% <u>Adam</u> FortranLib. 8.76% 0.92% all 47.03% 17.89% table 1.15x dam's sqrt: 4.54x x all 45.03% table 1.15x dam's sqrt: 4.54x x all 47.03% table 1.15x table 1.15x ta	 Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li>	ail cumtime percall filename:lineno(function) 0002 166.80 0.002 (method 'dot' of 'numpy.ndarray' objects) 0011 104.47 0.001 optimizers/adam.py:53(adam_kro2) 0000 440.19 0.002 function_node.py:201(apply) 0002 292.06 0.024 variable.py:968(_backward_main) 0000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN-1.3.0 and dit to the K computer users. and dit to the K computer users. xy of ChainerMN on the K computer is ment conditions: MNIST sample (unit=1000, 20 → 1epoch=600iter). n it can be measured even using 600 proc. or e must take care of recurrence and not of results. swell up to about 200 processes for this
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]: • • Using SSL II thread parallel BLAS in nu • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • O.04% → 0.92% → 17.89% (490x) Performance tuning of Chainer etapsed[s] 10,000.0 0	3]: $\frac{1}{4,096.0}$ $\frac{1}{1,384.0}$ $\frac{1}{4,096.0}$ $\frac{1}{1,15x}$ 1	 Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li>	allcumtimepercallfilename:lineno(function).002166.800.002 (method 'dot' of 'numpy.ndarray' objects).001104.470.001 optimizers/adam.py:53(adam_kro2).000440.190.002 function_node.py:201(apply).000292.060.024 variable.py:968(_backward_main).00021.160.000 numpy/ctypeslib.py:196(from_param)zation by the ChainerMNtried to install ChainerMN-1.3.0 andtried to install ChainerMN-1.3.0 andti to the K computer users000computer users000to be measured even using 600 proc. ore must take care of recurrence andcomputer. and.01can be measured even using 600 proc. ore must take care of recurrence and.01care of recurrence and
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]: • • Using SSL II thread parallel BLAS in nu • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • O.04%→0.92%→17.89% (490x) Performance tuning of Chainer elapsed[s] on the K computer 10,000.0 4552	3]: all $\frac{1}{4,096.0}$ original $\frac{1}{2,76\%}$ 0.04% underflow Kfast(Kns) 9.31% 0.19% GEMM thread 38.81% 0.48% Adam FortranLib. 8.76% 0.92% all 47.03% 17.89% 17.89% 47.03% 17.89% 47.03% 17.89% all 47.03% 17.89%	 Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li>	ail cumtime percall filename:lineno(function) 0002 166.80 0.002 (method 'dot' of 'numpy.ndarray' objects} 001 104.47 0.001 optimizers/adam.py:53(adam_kro2) 0000 440.19 0.002 function_node.py:201(apply) 0000 292.06 0.024 variable.py:968(_backward_main) 0000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN tried to install ChainerMN-1.3.0 and di it to the K computer users. cy of ChainerMN on the K computer is ment conditions: MNIST sample (unit=1000, 20 → 1epoch=600iter). n n it can be measured even using 600 proc. or e e must take care of recurrence and ncy of results. swell up to about 200 processes for this ample problem.
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]: • • Using SSL II thread parallel BLAS in nu • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • O.04%→0.92%→17.89% (490x) Performance tuning of Chainer etapsed[s] 10,000.0 6552	3]: $\frac{1}{4,096.0}$ $\frac{1}{1,384.0}$ $\frac{1}{4,096.0}$ $\frac{1}{1,15x}$ 1	 Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li>	ail cumtime percall filename:lineno(function) 0002 166.80 0.002 (method 'dot' of 'numpy.ndarray' objects} 001 104.47 0.001 optimizers/adam.py:53(adam_kro2) 0000 440.19 0.002 function_node.py:201(apply) 0000 292.06 0.024 variable.py:968(_backward_main) 0000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN-1.3.0 and tried to install ChainerMN-1.3.0 and di it to the K computer users. cy of ChainerMN on the K computer is ment conditions: MNIST sample (unit=1000, 20 → 1epoch=600iter). n it can be measured even using 600 proc. or e must take care of recurrence and cost distribution of Chainer
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]: • • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • O.04%→0.92%→17.89% (490x) Performance tuning of Chainer elapsed[s] on the K computer 10,000.0 - 6,000.0 -	all 47.03% 0.04% all 47.03% 0.19% GEMM thread 38.81% 0.48% Adam FortranLib. 8.76% 0.929% all 47.03% 17.89% umpy.dot: 1.15x dam's sqrt: 4.54x the sqrt and floating point underflow Scalability of Chainer 16,384.0 all 47.03% 0.48% on the K computer 1,384.0 ChainerMN ChainerMN ChainerMN Chainer	 Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li>	ail cumtime percall filename:lineno(function) 0002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} 001 104.47 0.001 optimizers/adam.py:53(adam_kro2) 0000 440.19 0.002 function_node.py:201(apply) 0002 292.06 0.024 variable.py:968(_backward_main) 0000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN-1.3.0 and the K computer users. tried to install ChainerMN-1.3.0 and the K computer users. zy of ChainerMN on the K computer is ment conditions: MNIST sample (unit=1000, 20 → 1epoch=600iter). n it can be measured even using 600 proc. or e must take care of recurrence and ncy of results. swell up to about 200 processes for this ample problem. Cost distribution of Chainer on the K computer
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]: • • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • O.04%→0.92%→17.89% (490x) Performance tuning of Chainer elapsed[s] on the K computer 10,000.0 48624	all 47.03% 0.04% all 47.03% 0.19% GEMM thread 38.81% 0.48% Adam FortranLib. 8.76% 0.929% all 47.03% 17.89% umpy.dot: 1.15x dam's sqrt: 4.54x the sqrt and floating point underflow Scalability of Chainer 16,384.0 all 47.03% 0.48% on the K computer 1,384.0 ChainerMN ChainerMN ChainerMN Chainer	 Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li>	ail cumtime percall filename:lineno(function) 0002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} 001 104.47 0.001 optimizers/adam.py:53(adam_kro2) 0000 440.19 0.002 function_node.py:201(apply) 0000 292.06 0.024 variable.py:968(_backward_main) 0000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN-1.3.0 and tried to install ChainerMN-1.3.0 and di it to the K computer users. ty of ChainerMN on the K computer is ment conditions: MNIST sample (unit=1000, 20 → 1epoch=600iter). n it can be measured even using 600 proc. or e must take care of recurrence and ncy of results. cost distribution of Chainer on the K computer
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]: • • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • O.04%→0.92%→17.89% (490x) Performance tuning of Chainer elapsed[s] on the K computer 10,000.0 - 6,000.0 -	3]: all 47.03% 0.04% Underflow Kfast(Kns) 9.31% 0.19% GEMM thread 38.81% 0.48% Adam FortranLib. 8.76% 0.92% all 47.03% 17.89% umpy.dot: 1.15x dam's sqrt: 4.54x c umpy.dot: 7.76%→38.81%→47.03% f dam's sqrt and floating point underflow scalability of Chainer 16,384.0 4,096.0 1,024.0 ChainerMN ChainerMN Chainer	 Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li>	ail cumtime percall filename:lineno(function) 0002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} 001 104.47 0.001 optimizers/adam.py:53(adam.kro2) 0000 440.19 0.002 function_node.py:201(apply) 0000 292.06 0.024 variable.py:968(_backward_main) 0000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN tried to install ChainerMN-1.3.0 and di it to the K computer users. zy of ChainerMN on the K computer is ment conditions: MNIST sample (unit=1000, 20 → 1epoch=600iter). n it can be measured even using 600 proc. or e must take care of recurrence and ncy of results. swell up to about 200 processes for this ample problem.
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]: • • Using of SWPL by Fortran library for Ac • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • 0.04%→0.92%→17.89% (490x) Performance tuning of Chainer * * * 0 nthe K computer * * * * * * * * * * * * * * * * * * * * * * * * </td <td>$3]: \frac{1}{4,096.0} \frac{1}{4,096.0} \frac{1}{256.0} \frac{1}{256.0} \frac{1}{256.0} \frac{1}{1,024.0} \frac{1}{256.0} \frac{1}{256.0} \frac{1}{1,024.0} \frac{1}{256.0} \frac{1}{1,024.0} \frac{1}{1,0$</td> <td> Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li></td> <td>ail cumtime percall filename:lineno(function) 0002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} 001 104.47 0.001 optimizers/adam.py:53(adam_kro2) 0000 440.19 0.002 function_node.py:201(apply) 0000 292.06 0.024 variable.py:968(_backward_main) 0000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN-1.3.0 and tried to install ChainerMN-1.3.0 and tried to the K computer users. try of ChainerMN on the K computer is ment conditions: MNIST sample (unit=1000, 20 → 1epoch=600iter). to about 200 processes for this ample problem. Cost distribution of Chainer on the K computer addam</td>	$3]: \frac{1}{4,096.0} \frac{1}{4,096.0} \frac{1}{256.0} \frac{1}{256.0} \frac{1}{256.0} \frac{1}{1,024.0} \frac{1}{256.0} \frac{1}{256.0} \frac{1}{1,024.0} \frac{1}{256.0} \frac{1}{1,024.0} \frac{1}{1,0$	 Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li>	ail cumtime percall filename:lineno(function) 0002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} 001 104.47 0.001 optimizers/adam.py:53(adam_kro2) 0000 440.19 0.002 function_node.py:201(apply) 0000 292.06 0.024 variable.py:968(_backward_main) 0000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN-1.3.0 and tried to install ChainerMN-1.3.0 and tried to the K computer users. try of ChainerMN on the K computer is ment conditions: MNIST sample (unit=1000, 20 → 1epoch=600iter). to about 200 processes for this ample problem. Cost distribution of Chainer on the K computer addam
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]: • • Using SSL II thread parallel BLAS in nu • Using SSL II thread parallel BLAS in nu • Using SSL II thread parallel BLAS in nu • Using SSL Yet by Fortran library for Ac • 0.04% → 0.92% → 17.89% (490x) Performance tuning of Chainer • • 0.04% → 0.92% → 17.89% (490x) • • • • • • • • • • • • • • • • • • • • •	$3]: \frac{1}{4,096.0} \frac{1}{64.0} \frac{1}{64.0} \frac{1}{1,024.0} \frac{1}{256.0} \frac{1}{64.0} \frac{1}{1,024.0} \frac{1}{256.0} \frac{1}{2} \frac{1}{$	 Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li>	ail cumtime percall filename:lineno(function) 0002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} 001 104.47 0.001 optimizers/adam.py:53(adam_kro2) 0000 440.19 0.002 function_node.py:201(apply) 0000 292.06 0.024 variable.py:968(_backward_main) 0000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN-1.3.0 and the K computer users. tried to install ChainerMN-1.3.0 and the K computer users. zy of ChainerMN on the K computer is ment conditions: MNIST sample (unit=1000, 20 → 1epoch=600iter). n it can be measured even using 600 proc. or e must take care of recurrence and ney of results. swell up to about 200 processes for this ample problem. Cost distribution of Chainer on the K computer mappe* 0.004 the K computer 0.004 the K computer mappe % 0.005 the K computer 0.007 the K computer mappe % 0.004 the K computer 0.007 the K computer mappe % 0.002 the K computer 0.008 the K computer 0.000 0.009 the K computer 0.000 the K computer 0.000 0.000 the K computer 0.000 the K computer 0.000 0.
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]: • • Using SSL II thread parallel BLAS in nu • Using SSL II thread parallel BLAS in nu • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • 0.04% → 0.92% → 17.89% (490x) Performance tuning of Chainer elapsed[s] • 10,000.0 • • 936.9 • • • 0.04% • • • • • • • • • • • • • •	$3]: \frac{1}{4,096.0} \frac{1}{64.0} \frac{1}{64.0} \frac{1}{1,024.0} \frac{1}{256.0} \frac{1}{64.0} \frac{1}{1,024.0} \frac{1}{256.0} \frac{1}{2} \frac{1}{$	 Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li>	ail cumtime percall filename:lineno(function) 0002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} 001 104.47 0.001 optimizers/adam.py:53(adam_kro2) 0000 440.19 0.002 function_node.py:201(apply) 0000 292.06 0.024 variable.py:968(_backward_main) 0000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN-1.3.0 and tried to install ChainerMN-1.3.0 and tried to the K computer users. try of ChainerMN on the K computer is ment conditions: MNIST sample (unit=1000, 20 → 1epoch=600iter). to about 200 processes for this ample problem. Cost distribution of Chainer on the K computer addam
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]: • • Using SSL II thread parallel BLAS in nu • Using SSL II thread parallel BLAS in nu • Using SSL II thread parallel BLAS in nu • Using SSL Yet by Fortran library for Ac • 0.04% → 0.92% → 17.89% (490x) Performance tuning of Chainer elapsed[s] on the K computer 12,000.0 138.9 0,000.0 138.9 0,000.0 138.9 0,000.0 138.9 14,400.0 14.3 14,400.0 14.3 0.0 14.89.4	$3]: \frac{1}{1024.0}	 Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li>	ail cumtime percall filename:lineno(function) 0002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} 001 104.47 0.001 optimizers/adam.py:53(adam_kro2) 0000 440.19 0.002 function_node.py:201(apply) 0000 292.06 0.024 variable.py:968(_backward_main) 0000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN-1.3.0 and the K computer users. tried to install ChainerMN-1.3.0 and the K computer users. zy of ChainerMN on the K computer is ment conditions: MNIST sample (unit=1000, 20 → 1epoch=600iter). n it can be measured even using 600 proc. or e must take care of recurrence and ney of results. swell up to about 200 processes for this ample problem. Cost distribution of Chainer on the K computer mappe* 0.004 the K computer 0.004 the K computer mappe % 0.005 the K computer 0.007 the K computer mappe % 0.004 the K computer 0.007 the K computer mappe % 0.002 the K computer 0.008 the K computer 0.000 0.009 the K computer 0.000 the K computer 0.000 0.000 the K computer 0.000 the K computer 0.000 0.
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3] • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Adam • floating point underflow control: 3.38> • Total speedup: 16.2x - Efficiency improvements [Table 4]: • • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Adam • Using of SWPL by Fortran library for Adam • Using of SWPL by Fortran library for Adam • 0.04%→0.92%→17.89% (490x) Performance tuning of Chainer elapsed[s] on the K computer 10,000.0 3222 • 336.9 • 136.9 • 41.3 • 336.9 • 438.8 • 41.3 • 14.9 • 990.2 • 41.3	3]: all 10000000000000000000000000000000000	 Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li>	ail cumtime percall filename:lineno(function) 0002 166.80 0.002 {method 'dot' of 'numpy.ndarray' objects} 001 104.47 0.001 optimizers/adam.py:53(adam_kro2) 0000 440.19 0.002 function_node.py:201(apply) 0002 292.06 0.024 variable.py:968(_backward_main) 0000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN-1.3.0 and ti to the K computer users. tried to install ChainerMN-1.3.0 and ti to the K computer users. try of ChainerMN on the K computer is ment conditions: MNIST sample (unit=1000, 20) → 1epoch=600iter). n it can be measured even using 600 proc. or e must take care of recurrence and ncy of results. swell up to about 200 processes for this ample problem. Margin 000 001 002 003 004 005 005 005 000 000 000 000 000 000 000
Adam FortranLib. 1,052.6 780.7 439.6 2,272.9 all 194.9 41.3 399.0 635.2 5. Result - Speedups by each tuning step [Fig.3 • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • floating point underflow control: 3.38x • Total speedup: 16.2x - Efficiency improvements [Table 4]: • • Using SSL II thread parallel BLAS in nu • Using SSL II thread parallel BLAS in nu • Using of SWPL by Fortran library for Ac • 0.04%→0.92%→17.89% (490x) Performance tuning of Chainer elapsed[s] on the K computer 10,000.0 436.9 • 4,000.0 326.9 • 0.0 438.6 41.3 0.0 326.9 • • 198.9 •	$3]: \frac{1}{1024.0}	 Tuned Chainer Ver.4.4.0 p ncalls tottime percent of the /li>	ail cumtime percall filename:lineno(function) 0002 166.80 0.002 (method 'dot' of 'numpy.ndarray' objects} 001 104.47 0.001 optimizers/adam.py:53(adam_kro2) 0000 440.19 0.002 function_node.py:201(apply) 0002 292.06 0.024 variable.py:968(_backward_main) 0000 21.16 0.000 numpy/ctypeslib.py:196(from_param) zation by the ChainerMN-1.3.0 and ti to the K computer users. tried to install ChainerMN-1.3.0 and ti to the K computer users. try of ChainerMN on the K computer is ment conditions: MNIST sample (unit=1000, 20 → 1epoch=600iter). n it can be measured even using 600 proc. or e must take care of recurrence and ncy of results. swell up to about 200 processes for this ample problem. Takes 000 101 102 103 104 105 105 104 105 105 106 107 108 108 109

There are some limitations on the use of Chainer on the K computer. It is necessary to prepare the learning data beforehand and to stage-in the data to an appropriate storage system. Moreover, since Python is in the shared storage, it takes time to load the library. However, it was confirmed that we can use the K computer for deep learning sufficiently as well as GPU.



ISC 2019 Messe Frankfurt, June

 1proc. original
 1proc. all
 48proc. original

 Fig.5 The change of cost distribution by the parallelization effect.

194.9

20.6

20%

10%

0%

1,189.1