BLAZING TEXT

Scaling and Accelerating Word2Vec using Multiple GPUs

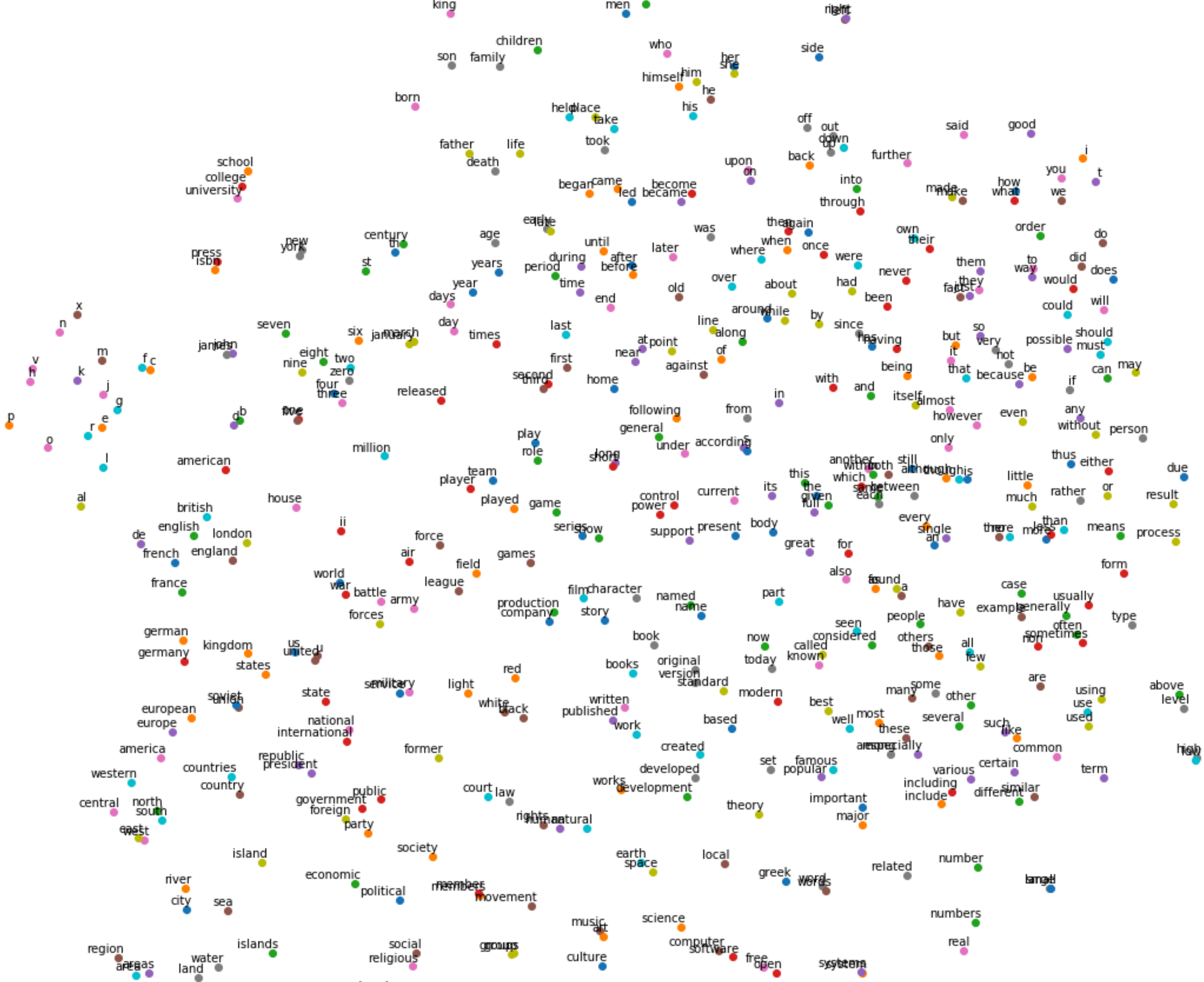
Saurabh Gupta
Applied Scientist, Amazon AI

OVERVIEW

- What is Word2Vec?
- Need for speed - Can deep learning frameworks help?
- Hardware view: NVIDIA GPU architecture
- Software view: CUDA C/C++
- BlazingText: GPU acceleration and performance
- Future work

WORD2VEC

- Dense low-dimensional embedding space such that the geometry of resulting vectors captures word semantic similarity
- vec(“King”) - vec(“Queen”) + vec(“Woman”) = vec(“Man”)
- This idea has enabled many Natural Language Processing (NLP) algorithms to achieve better performance - Machine Translation, POS tagging, NER, Sentiment analysis.
- How to train? Predict surrounding words in a window of length *m* of every word.

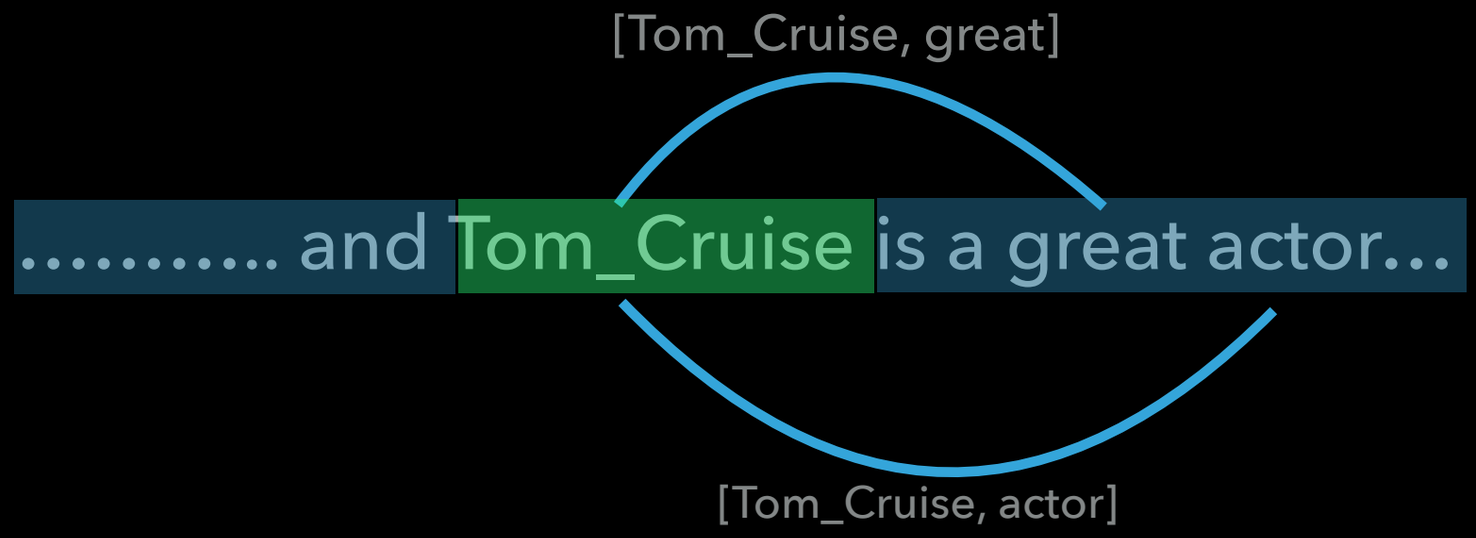


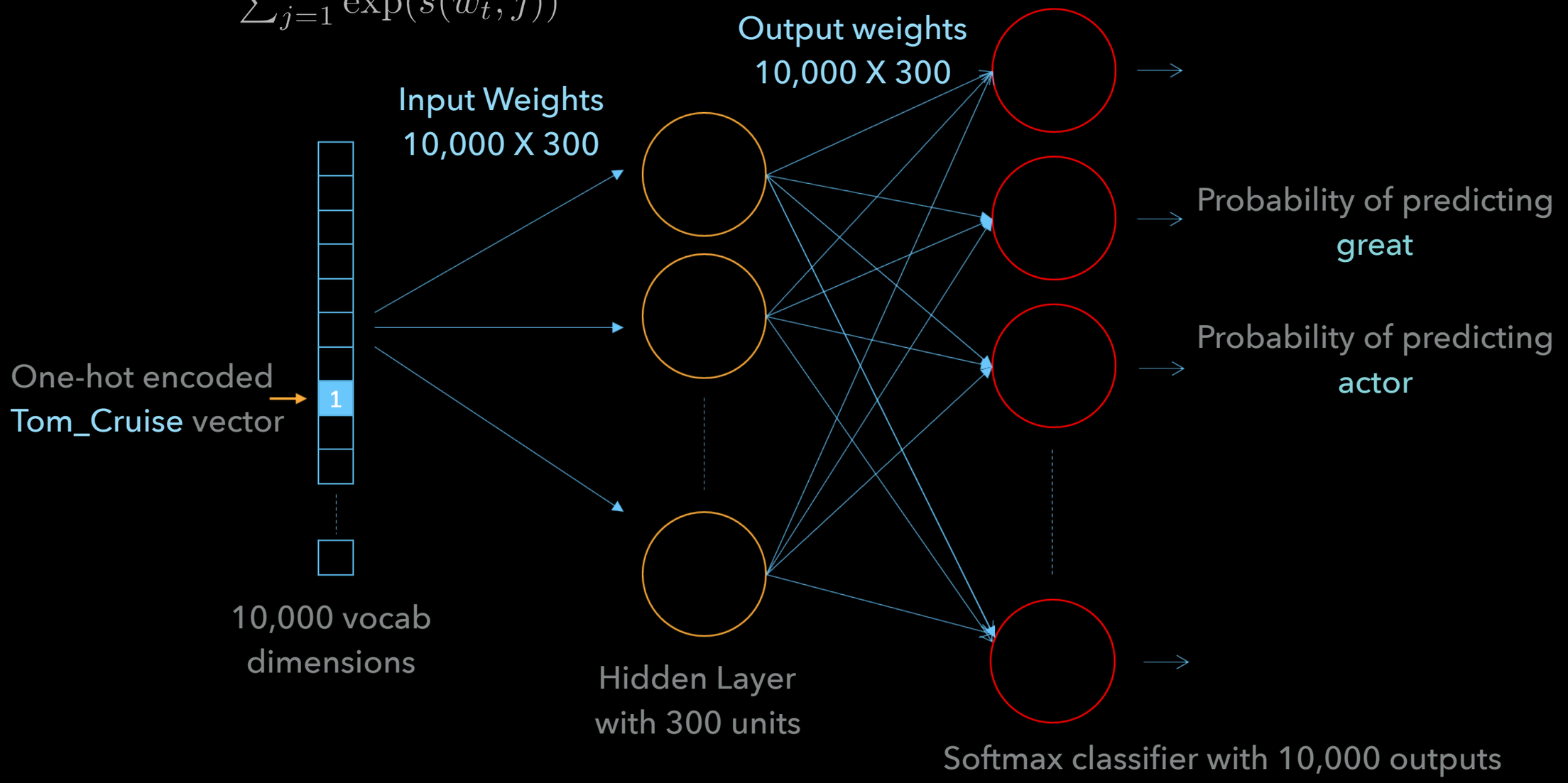
king
men
children
son
family
who
him
himself
he
side
born
his
take
took
off
out
down
said
good
father
life
death
school
college
university
upon
on
back
further
into
through
i
you
t
how
what
we
began
came
led
become
became
age
century
new
york
press
isbn
st
years
period
during
until
after
before
later
was
again
when
once
were
where
order
do
own
their
never
them
to
way
did
does
year
days
time
end
old
over
about
had
been
would
could
will
x
n
seven
six
january
march
day
times
last
line
along
around
while
by
since
having
but
so
very
possible
should
must
james
john
eight
two
zero
nine
four
three
first
second
third
near
point
against
of
being
it
that
not
because
be
can
may
if
m
f
c
v
h
k
j
g
p
e
r
o
l
b
released
home
in
with
and
itself
almost
however
even
any
without
person
million
play
role
following
general
under
from
according
only
american
player
team
played
game
long
short
control
power
current
this
the
which
its
given
full
another
both
each
between
still
although
little
much
rather
or
thus
either
due
result
al
british
english
london
de
french
england
house
ii
force
air
field
games
series
show
support
present
body
great
for
every
single
an
than
more
less
means
process
form
france
world
war
battle
army
forces
league
production
company
film
character
story
named
name
part
also
found
a
have
case
example
generally
usually
type
people
german
germany
kingdom
us
united
states
book
now
today
seen
considered
called
known
others
those
all
few
often
sometimes
non
books
original
version
standard
red
light
service
military
soviet
union
state
european
europe
white
black
written
published
work
modern
many
some
other
best
most
well
these
several
such
like
are
using
use
used
above
level
national
international
america
western
countries
country
republic
president
former
created
based
set
famous
popular
especially
common
certain
various
including
include
different
similar
term
high
low
central
north
south
east
west
government
foreign
public
party
court
law
rights
natural
works
development
developed
theory
important
major
island
society
earth
space
local
greek
word
words
related
number
river
economic
political
members
member
movement
city
sea
music
art
science
numbers
region
area
areas
water
land
islands
social
religious
groups
group
culture
computer
software
free
open
systems
system
real
large
small

WORD2VEC INTUITION

Given a sentence -
Try to maximize the probability of predicting context words.

$$p(w_c|w_t) = \frac{\exp(s(w_t, w_c))}{\sum_{j=1}^{W} \exp(s(w_t, j))}$$





WORD2VEC INTUITION

Finally, the input weight matrix (10,000 X 300) is what we are interested in. It is nothing but a word vector lookup table!

However, it is very inefficient to learn the softmax weights (huge summation in the denominator!). The neural network has a tremendous number of weights, all of which would be updated slightly by every one of our billions of training samples!

So, Mikolov et al introduced negative sampling, according to which the following objective function should be maximized:

$$\log \sigma(w^T.c) + \sum_{neg=1}^{5} \log \sigma(-w^T.w_{neg})$$

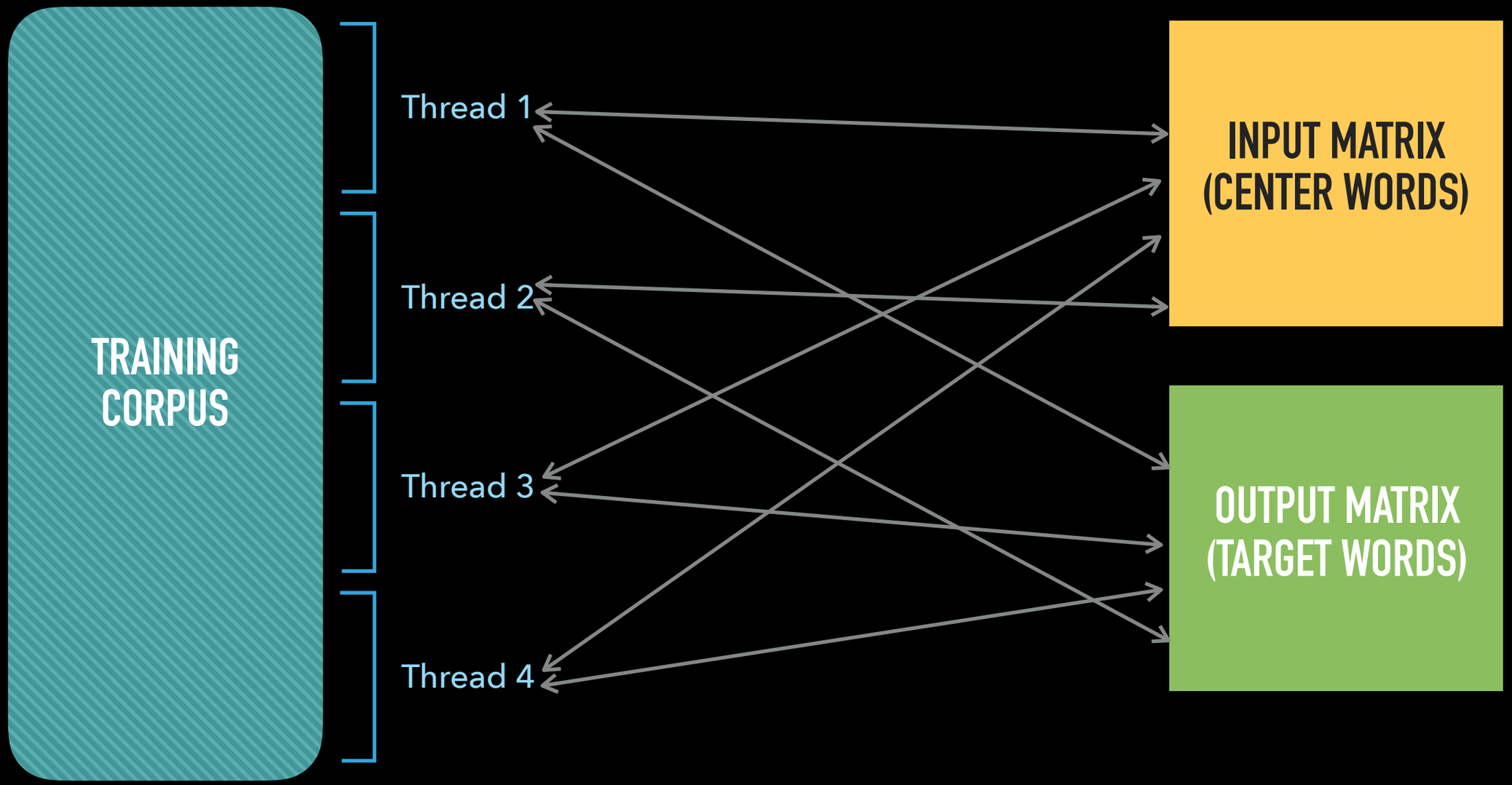
Tom_Cruise

actor

Any random word vector not in Tom_Cruise's context

Note: *w* and *c* come from different weight matrices. Think of family of LR classifiers!

WORD2VEC ACCELERATION USING ASYNC SGD ON CPU



NEED FOR SPEED! USE GPU?

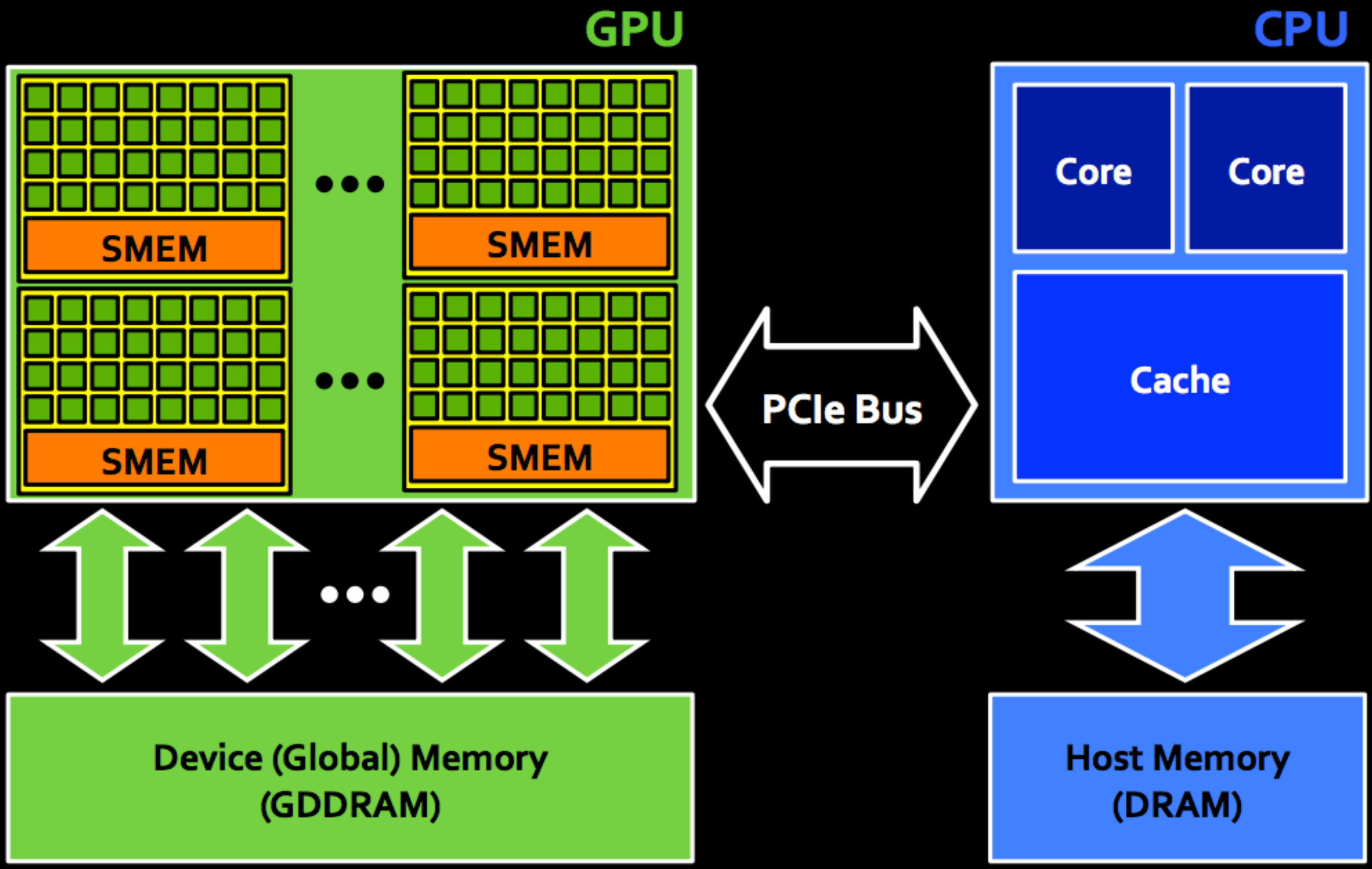
- Many downstream NLP applications use Word2Vec to initialize word embeddings.
- Datasets can be of the order of several GBs, on which CPU Word2Vec can take several hours or even days.
- GPU to the rescue - Use TensorFlow, MXNet, PyTorch etc ?
- These frameworks not very suitable for this application (Hard to beat CPU implementations - Gensim, FastText):
  - Network is not that deep. Do gradient math by hand and write your own kernels.
  - SGD with batch size = 1 works the best. Large batches affect convergence significantly - defeats the purpose of using a deep learning framework
  - Data Iteration is compute intensive. Very slow in Python due to GIL.
- Use CUDA APIs for a fine grained control over GPU parallelism!

HETEROGENEOUS PARALLEL COMPUTING

- CPU optimized for fast single-thread execution
  - Cores designed to execute 1 thread or 2 threads concurrently
  - Large caches attempt to hide DRAM access times
  - Cores optimized for low-latency cache accesses
  - Complex control-logic for speculation and out-of-order execution

- GPU optimized for high multi-thread throughput
  - Cores designed to execute many parallel threads concurrently
  - Cores optimized for data-parallel, throughput computation
  - Chips use extensive multithreading to tolerate DRAM access times

HARDWARE VIEW



CUDA EXECUTION MODEL

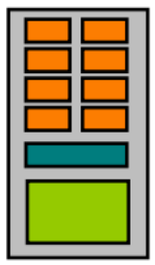
Software

Thread

Thread Block

Hardware

CUDA core



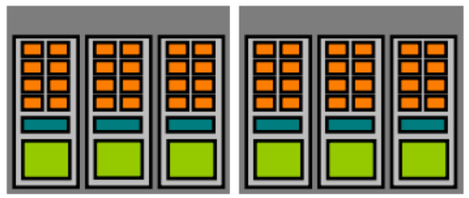
Multiprocessor

Threads are executed by CUDA cores

Thread blocks are executed on multiprocessors

- Thread blocks do not migrate
- Several concurrent thread blocks can reside on one multiprocessor - limited by multiprocessor resources (shared memory and register file)

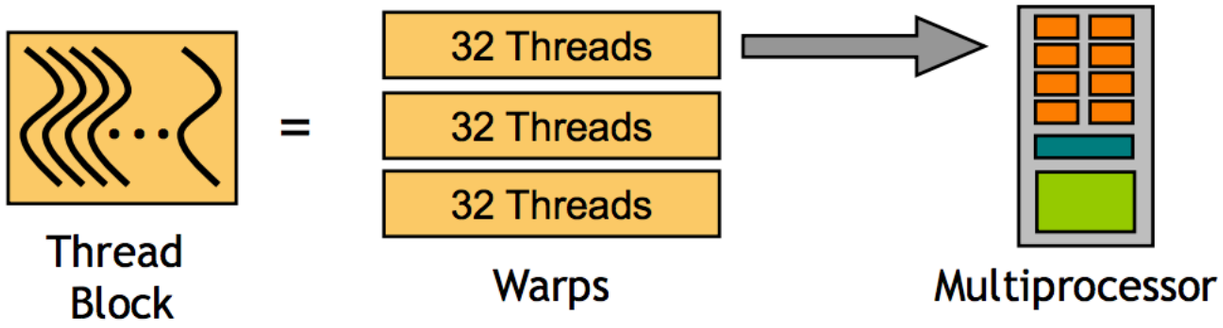
Grid



Device

A kernel is launched as a grid of thread blocks
Blocks and grids can be multi dimensional (x,y,z)

CUDA WARPS

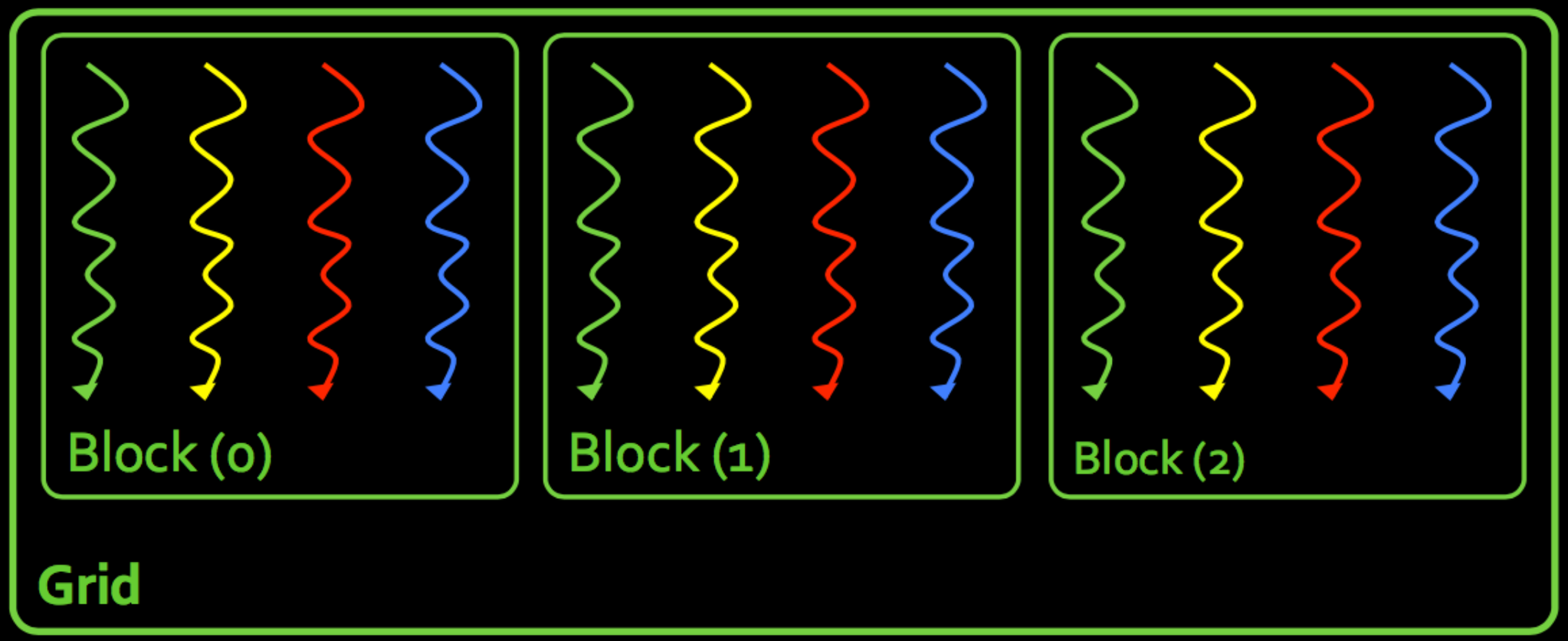


A thread block consists of one or more warps

A warp is executed physically in parallel (SIMD) on a multiprocessor

Currently all NVIDIA GPUs use a warp size of 32

BLOCK OF THREADS AND GRID OF BLOCKS



- Threads are grouped into blocks
- Blocks are grouped into a grid
- A kernel is executed as a grid of blocks of threads

BLOCKS ENABLE EFFICIENT COLLABORATION

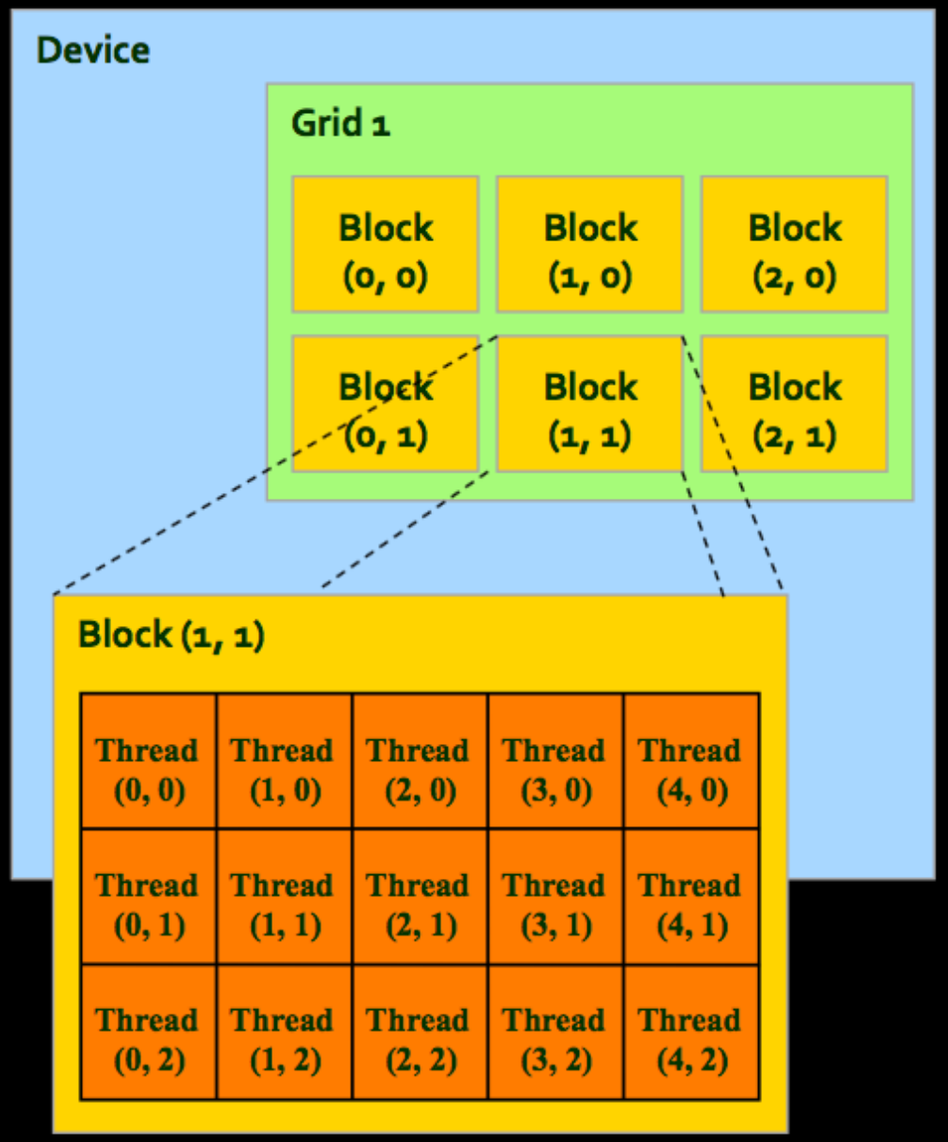
- Threads often need to collaborate
  - Cooperatively load/store common data sets
  - Share results or cooperate to produce a single result
  - Synchronize with each other

- Threads in the same block
  - Can communicate through shared and global memory
  - Can synchronize using fast synchronization hardware

- Threads in different blocks of the same grid
  - Cannot synchronize reliably
  - No guarantee that both threads are alive at the same time

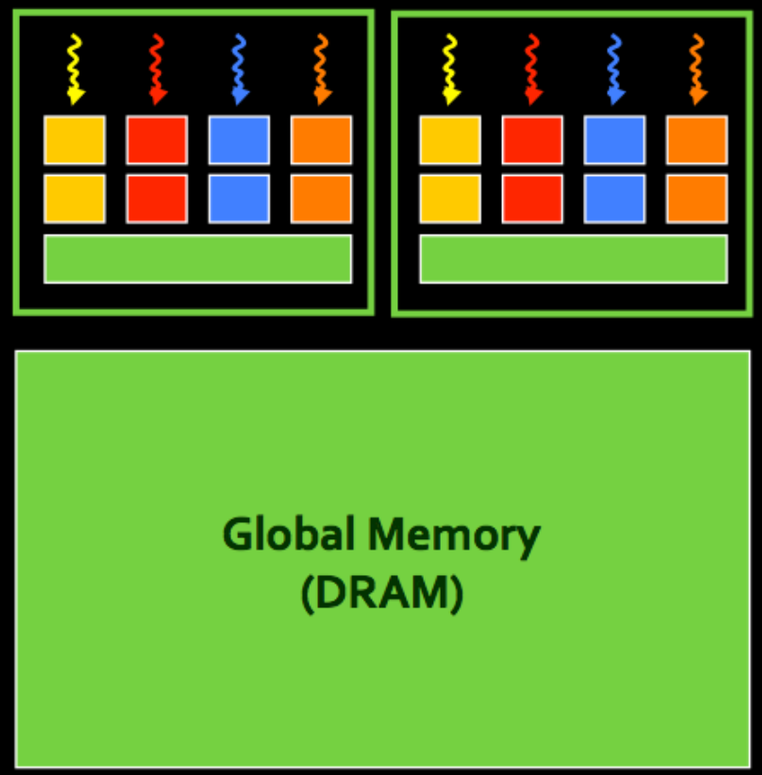
THREAD AND BLOCK ID AND DIMENSIONS

- Threads
  - 3D IDs, unique within a block
- Thread Blocks
  - 2D IDs, unique within a grid
- Dimensions set at launch
  - Can be unique for each grid
- Built-in variables
  - `threadIdx`, `blockIdx`
  - `blockDim`, `gridDim`
- Programmers usually select dimensions that simplify the mapping of the application data to CUDA threads



CUDA MEMORY HIERARCHY AND GLOBAL MEMORY

- Allocated explicitly by host (CPU) thread
- Scope: all threads of all kernels
- Data lifetime: determine by host (CPU) thread
  - cudaMalloc (void ** pointer, size_t nbytes)
  - cudaFree (void* pointer)
- Capacity: large (1-6GB)
- Latency: 400-800 cycles
- Bandwidth: 156 GB/s
  - Data access patterns will limit bandwidth achieved in practice
- Common uses
  - Staging data transfers to/from CPU
  - Staging data between kernel launches



BLAZING TEXT

Word2Vec Acceleration On GPU

GPU ACCELERATION CHALLENGES:

- Just like CPU, use n threads to break the file into n parts and do async SGD? Not that straightforward! Remember that a single GPU thread is much much slower than a single CPU thread!

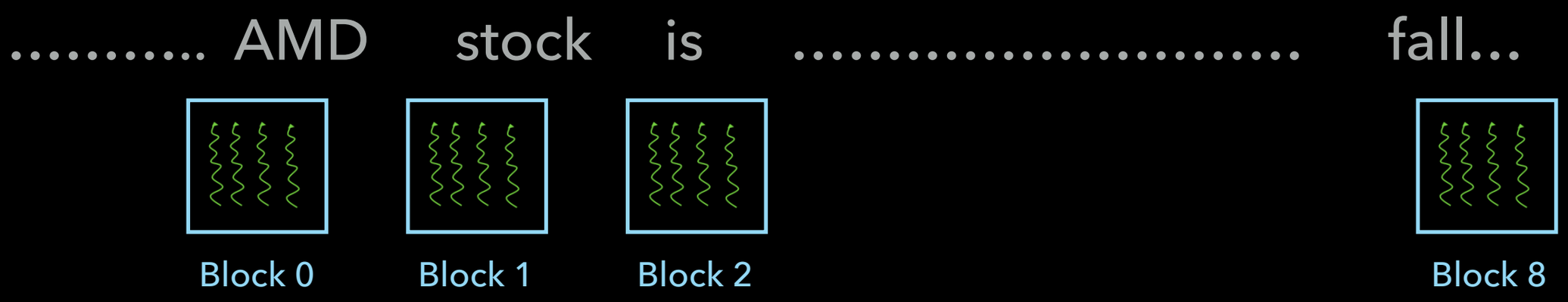
- Cannot assume that the file can reside in GPU memory (~12GB). Stream sentences from disk to GPU's DRAM? Transfer speed not that good. Batch streaming of sentences to GPU RAM to amortize cost of data transfer? Possible!

- Given the CUDA threading and memory model, use one GPU thread per center word? Probably no. Use ~100 threads per word if vector dim = 100

- Threads for vector dot products need to synchronize to calculate the sum (reduce operation). So one thread block per word? And have as many thread blocks as the MAX SENTENCE LENGTH?

- The above approach seems reasonable. But really?

APPROACH 1: MASSIVELY PARALLEL



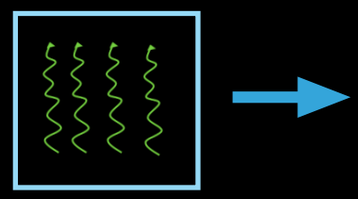
Although this approach is massively parallelized, it results in huge accuracy drop. When a window moves across a sentence, the vector of each word can be updated up to 2w times, where w is the size of the window. However, with this approach, due to the parallelism at word level, these updates might get lost.

Alternative approach:

Map a sentence to a CUDA block and each dimension of word vector to a thread. Each word in the sentence mapped to a block is processed sequentially. This approach might result in race conditions as well but to a lesser degree than the former approach.

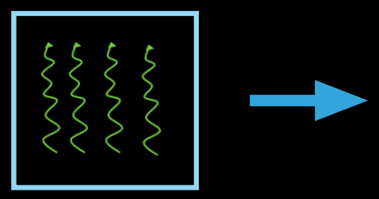
APPROACH 2: CONTROL EXCESSIVE PARALLELISM FOR ACCURACY

Sentence 0.....



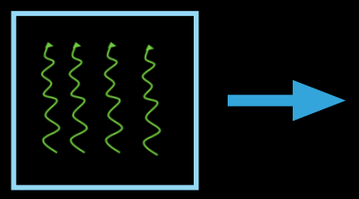
Block 0

Sentence 1.....



Block 1

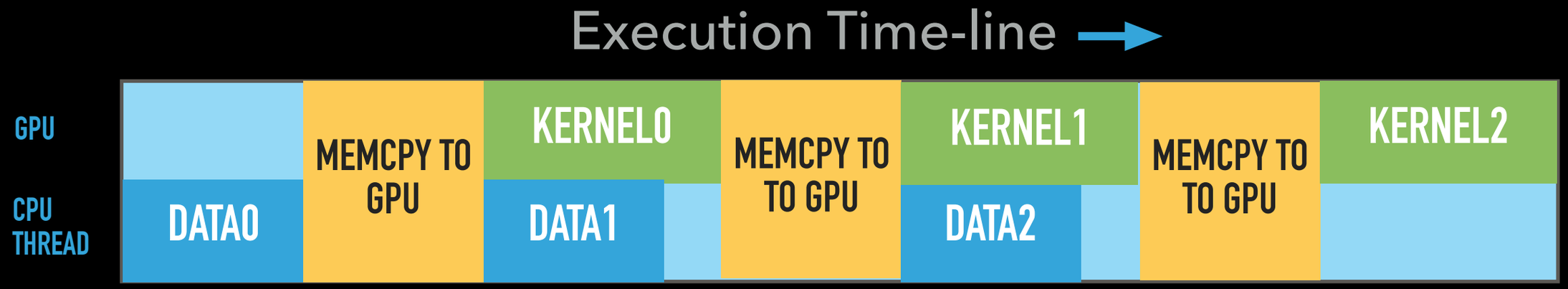
Sentence n.....



Block n

Batch sentences and transfer to GPU. Let each thread block process a sentence!
This approach results in 1.6 - 2x speedup over 8 threaded CPU while retaining the accuracy.
More implementation details can be found in the paper.

Data I/O can be the bottleneck too. Use CUDA streams.



SCALING TO MULTIPLE GPUS

- Use data parallelism
- The model parameters - Input and Output vectors for all the words in vocabulary, are replicated on each GPU
- Each device then independently processes the data partition it owns and updates its local model, periodically synchronizing the local model with all other N-1 GPUs.
- **Efficient model synchronization:** NVIDIA's NCCL library, which provides an AllReduce method that handles the peer-to-peer data transfer between the GPUs in an optimized way based on the topology of GPU network.
- Synchronization frequency?

SCALING TO MULTIPLE GPUS: THROUGHPUT

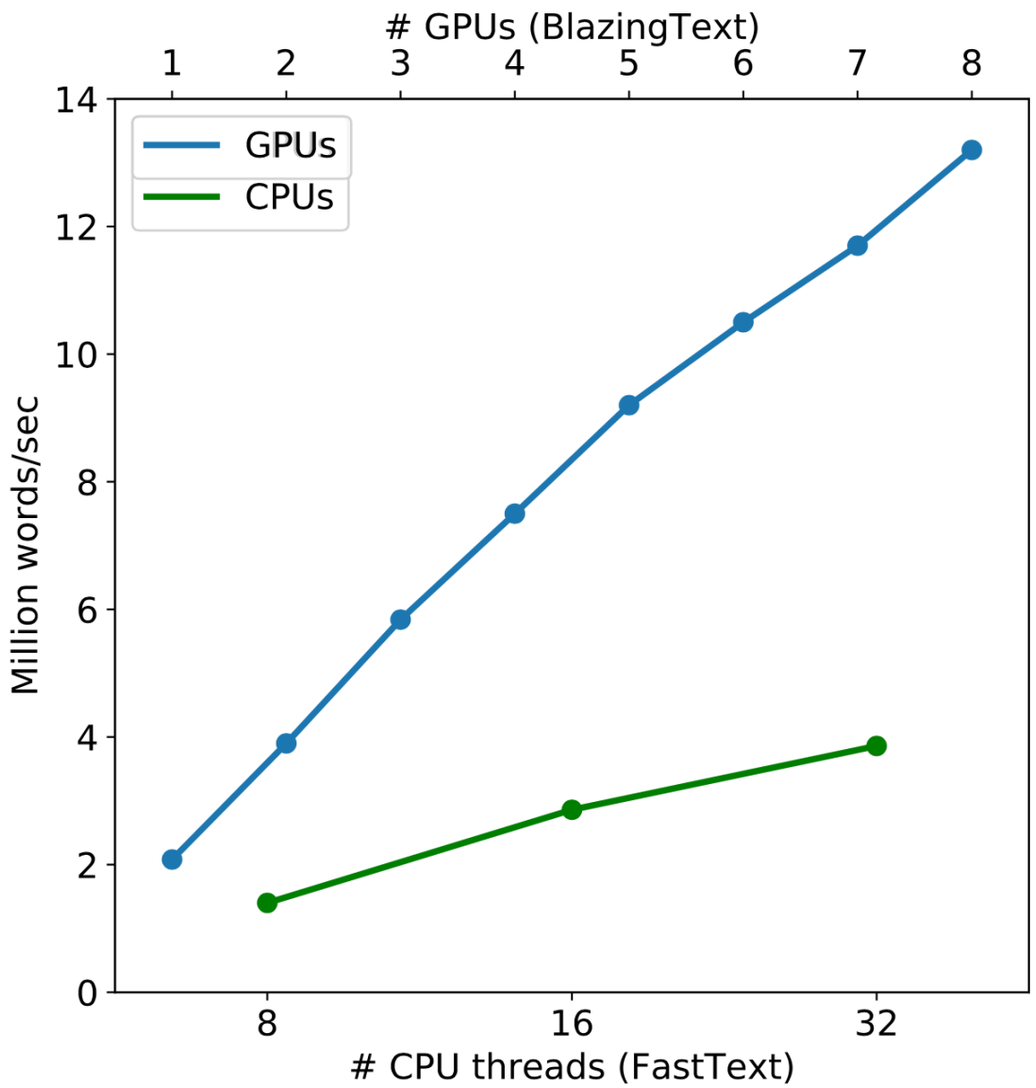


Figure 1: Skipgram throughput

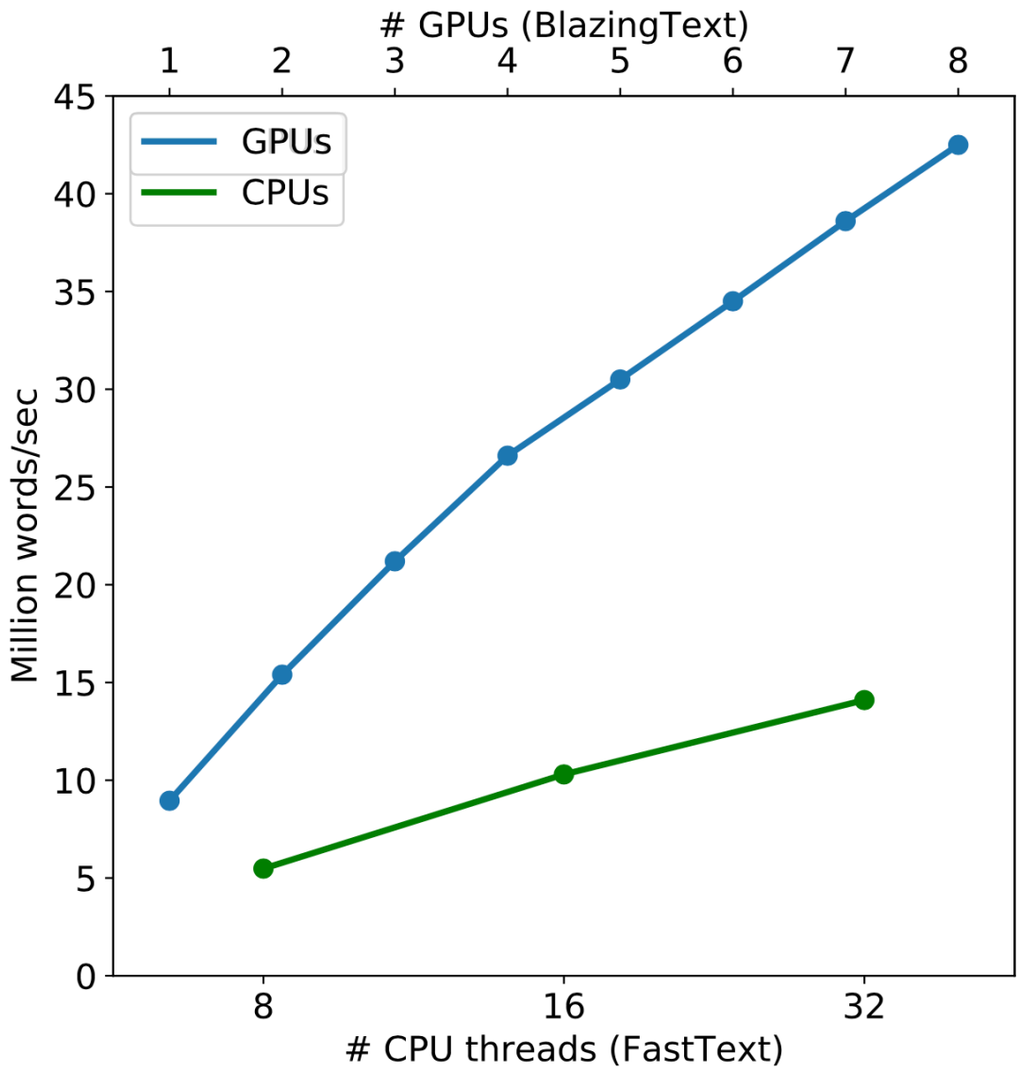


Figure 2: CBOW throughput

SCALING TO MULTIPLE GPUS: ACCURACY

Table 1: Spearman's rank correlation coefficient between model scores and human judgement on WS-353 dataset for word similarity. For word analogy task, we report the accuracies on Google analogy dataset.

Training Corpus			# GPUs (BlazingText)								# CPUs (FastText)		
			1	2	3	4	5	6	7	8	8	16	32
Text8	Similarity	Skipgram	.716	.713	.710	.707	.703	.699	.695	.694	.707	.706	.7
		CBOW	.711	.708	.705	.689	.683	.681	.679	.675	.694	.689	.69
	Analogy	Skipgram	.327	.324	.321	.311	.299	.297	.296	.295	.329	.330	.326
		CBOW	.321	.329	.32	.299	.295	.289	.285	.281	.323	.326	.325
1 Billion word benchmark	Similarity	Skipgram	.659	.658	.656	.653	.655	.659	.651	.650	.660	.659	.656
		CBOW	.609	.607	.599	.598	.601	.604	.598	.597	.610	.607	.608
	Analogy	Skipgram	.301	.305	.299	.299	.298	.297	.295	.289	.300	.302	.301
		CBOW	.299	.296	.295	.29	.289	.289	.287	.288	.311	.314	.312

FUTURE WORK

- Use Volta GPU. Available on AWS as P3 instances.
- Better synchronization frequency model - exploit the fast that model updates are linked to word frequency.
- Better learning rate scheduling.

THANK YOU!

QUESTIONS?