RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

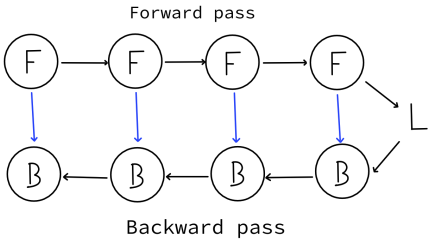
Inria

Pipeline Parallelism with Rematerialization
A scheduling perspective

GT Topal

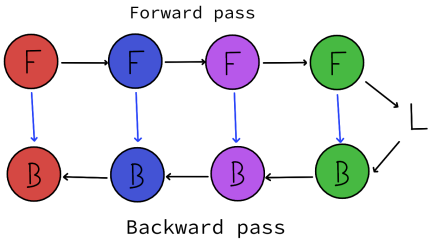
Adrien Aguila--Multner Olivier Beaumont Julia Gusak

Pipeline Parallelism for large LLMs



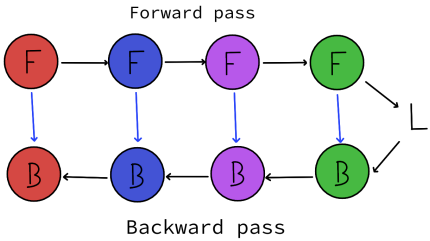
- Training process = same DAG many times
- Node = computation
  roughly F costs 1, B costs 2
- Edges = (long) data dependencies

Pipeline Parallelism for large LLMs



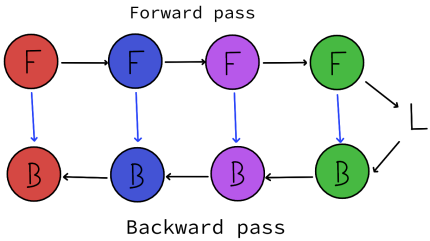
- Training process = same DAG many times
- Node = computation
- Edges = (long) data dependencies
- Pipeline parallelism: Split model into *stages*

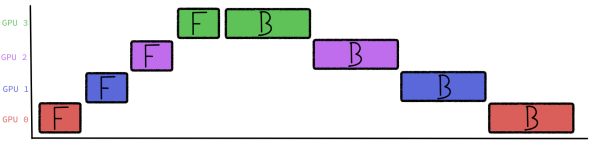
Pipeline Parallelism for large LLMs



- Training process = same DAG many times
- Node = computation
- Edges = (long) data dependencies
- Pipeline parallelism: Split model into *stages*
- Assign each stage to a different GPU
  - Around 128 layers for large LLMs
  - Identical layers (except possibly first and last) but different weights
  - load balancing is "easy" (without memory issues)

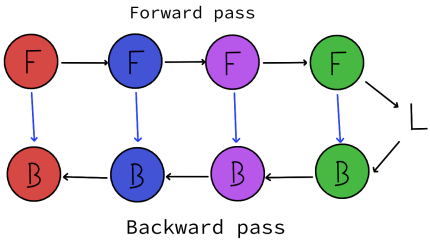
Pipeline Parallelism for large LLMs



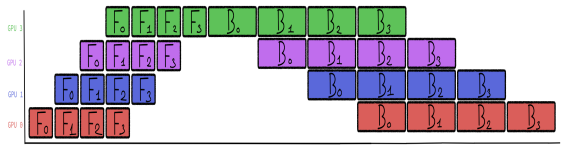


- Training process = same DAG many times
- Node = computation
- Edges = (long) data dependencies
- Pipeline parallelism: Split model into *stages*
- Assign each stage to a different GPU
- Granularity Issues:
  - Microbatch size: number of "sequences" grouped into one "F"
  - For large LLMs, a size of 1 is enough to keep a GPU happy !
  - Even more, we can use Tensor and Sequence Parallelisms with size 1

Pipeline Parallelism for large LLMs

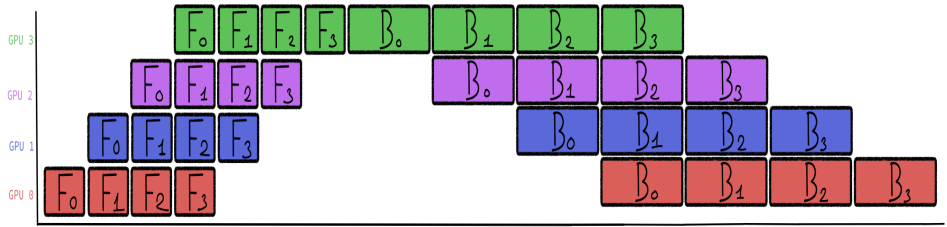


- Training process = same DAG many times
- Node = computation
- Edges = (long) data dependencies
- Pipeline parallelism: Split model into *stages*
- Assign each stage to a different GPU
- Granularity Issues:
- Also split data into micro-batches
  - Microbatch size: 1
  - Several (m) micro batches



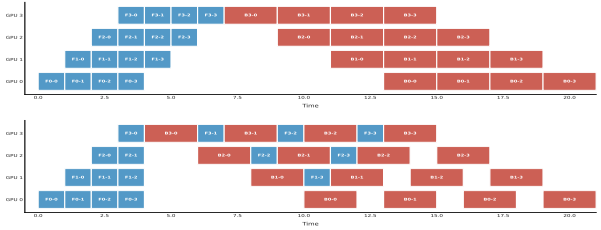
GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism, Huang et al. 2019

Pipeline Parallelism



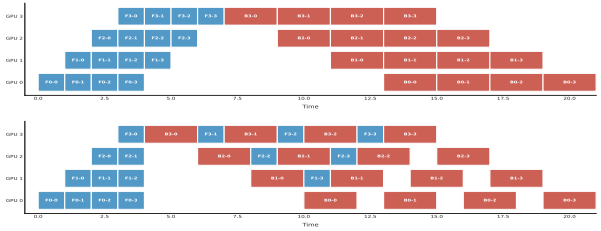
- Main Issues
- Scheduling:
  - Green F_0 before Green B_0 (and probably Green F_1)
  - the relative ordering Green F_1 and Green B_0 is free
- Memory Issues
  - Increasing the number of microbatches m increases efficiency
  - because the overall bubble size is independent of m
  - but you need to keep m outputs of F of size M_F instead of 1 !

Pipeline Parallelism - First solutions



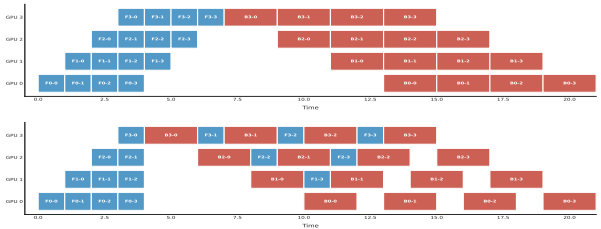
- 1F1B Scheduling:
  - Same makespan
  - But critical path is $(N-1)F + m(F+B) + (N-1)B$
  - Proof: trivial
    - Consider the first F and the last B on the GPU that owns the last part
    - $(N-1)F$ before the first F, $(N-1)B$ after the last B
    - $m(F+B)$ between them
- Memory Issues

Pipeline Parallelism - First solutions



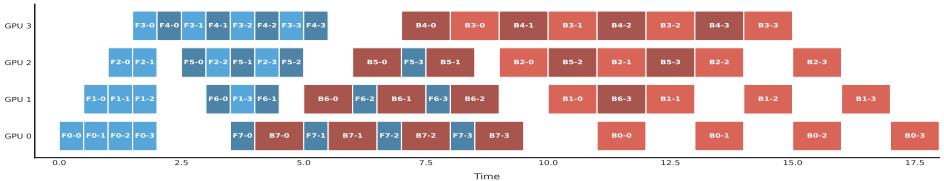
- Scheduling: critical path is $(N-1)F + m(F+B) + (N-1)B$
- Memory Issues :
  - Different memory peaks
  - B tasks free memory
  - In case of choice, we should prioritize an old B over a fresh F

Pipeline Parallelism - How to improve makespan ?



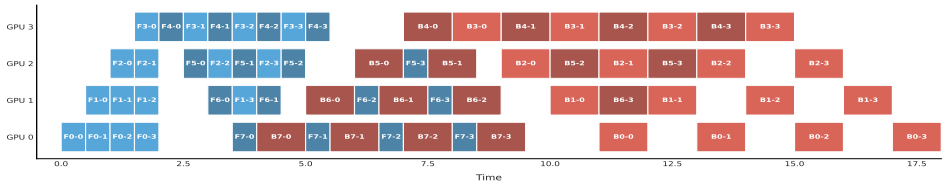
- The main issue was that GPU_0 works at the beginning and at the end
- while GPU_N works in the middle
- Solution: Multiple waves

Pipeline Parallelism - How to improve makespan ?



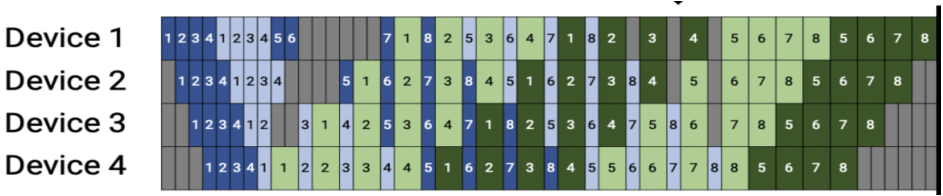
- The main issue was that GPU_0 works at the beginning and at the end
- while GPU_N works in the middle
- Solution: Multiple waves
  - Split the layers into $2N$ stages
  - Give 2 stages to each GPU (can be generalized to 4,...)
    - Stages i and $2N - i$: Hanayo Style
    - Stages i and $N + i$: Megatron Style

Pipeline Parallelism - Lower Bounds with multiple waves



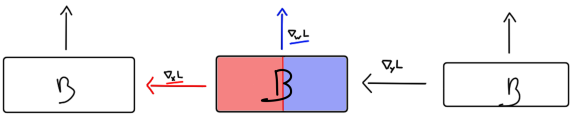
- Makespan is at least $(m + (N-1)/2)(F+B)$
  - Let us assume that GPU_k has layers f_k and l_k, $1 \leq f_k < l_k \leq 2N$.
  - Lemma: $\exists k,\ s.t.\ f_k \leq N$ (enumeration of small indexes)
  - Consider the first F and the last B **of layer** f_k on GPU_k
  - $(f_k - 1)F/2$ before the first, $(f_k - 1)B/2$ after the last and at least $2m(F+B)/2$ between them
  - $\Rightarrow$ makespan is larger than $(m + (f_k - 1)/2)(F+B)$
- this time, the bound is not tight for Hanayo (18/16.5): Idle time on the GPU that holds layer N

Pipeline Parallelism - Lower Bounds with multiple waves

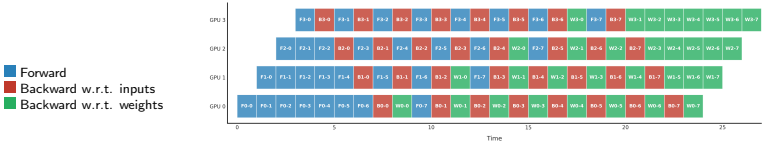


- Makespan is at least $(m + (N-1)/2)(F+B)$
  - Let us assume that GPU_k has layers f_k and l_k, $1 \leq f_k < l_k \leq 2N$.
  - Lemma: $\exists k,\ s.t.\ f_k \leq N$ (enumeration of small indexes)
  - Consider the first F and the last B **of layer** f_k on GPU_k
  - $(f_k - 1)F/2$ before the first, $(f_k - 1)B/2$ after the last and at least $2m(F+B)/2$ between them
  - $\Rightarrow$ makespan is larger than $(m + (f_k - 1)/2)(F+B)$
- this time, the bound is not tight for Hanayo (18/16.5): Idle time on the GPU that holds layer N
- but it is for Megatron (28.5/28.5)

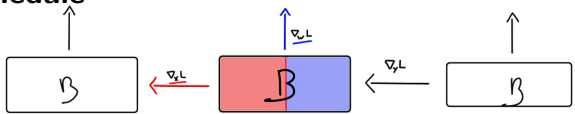
To go further, change the rules ! the Zero-Bubble trick & ZBH-2 Schedule



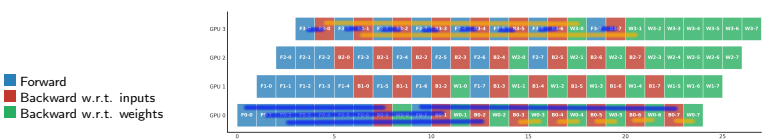
- Two very good ideas:
  - Decouple Backward Gradients (B) and Backward Weigths (W)
  - Change the definition of the makespan



To go further, change the rules ! the Zero-Bubble trick & ZBH-2 Schedule



- Two very good ideas:
  - Decouple Backward Gradients (B) and Backward Weigths (W)
  - Change the definition of the makespan



- But it makes Memory issues even worse !
  - Need to inject at least $m = 2N$ microbatches (N was enough for Hanayo and Megatron)
  - Memory Issues between F and B AND between B and W !

Pipeline Parallelism and Memory

- Pipeline parallelism has plenty of nice properties
  - It distributes well the weights among GPUs (each as a share of $1/N$)
  - With Zero-Bubble trick, parallel efficiency is very good
  - It only induces P2P communications with (relatively) small messages
- But memory remains an issue
  - To avoid bubbles an idle time, m must be large (typically $m = 2N$ for ZBH-2)
  - prevents to use a large number of GPUs for PP !
- Also limited by the size of your model (how much you can divide it)
- In what follows: what is the best schedule if you don't have memory (at all)

Pipeline parallelism when memory is too small !

- What can you do if you don't have enough memory ?

Pipeline parallelism when memory is too small !

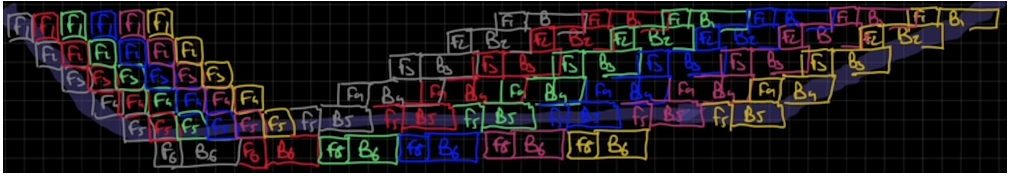
- What can you do if you don't have enough memory ?
  - One option is offloading (move from CPU memory to GPU memory)
  - Delete and then recompute the activations: re-materialization
  - both can be combined
- In what follows (no offloading)
  - How to find a good / optimal (i.e. both schedule / re-materialization policy) combination ?
  - What is the consequence of limited memory on lower bounds

Pipeline parallelism when memory is too small !

- What can you do if you don't have enough memory ?
  - One option is offloading (move from CPU memory to GPU memory)
  - Delete and then recompute the activations: re-materialization
  - both can be combined
- In what follows (no offloading)
  - How to find a good / optimal (i.e. both schedule / re-materialization policy) combination ?
  - What is the consequence of limited memory on lower bounds
- Two steps
  1. With almost no memory: just enough to store the activations of a single F
  2. With K memory slots for F activations

Pipeline parallelism when memory is very small - A good schedule !

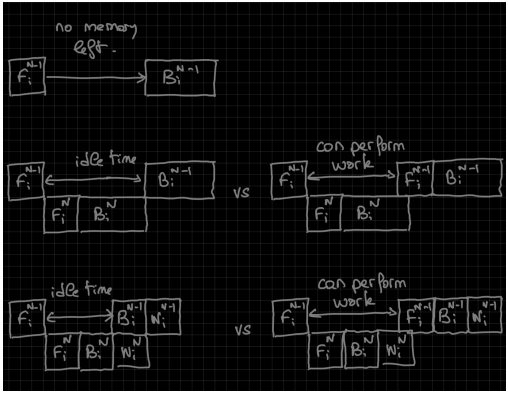
Proposed solution: Gpipe like schedule (except on last processor) with systematic recomputation



- Proving correctness is easy
  - F activations are systematically deleted unless the associated B immediately follows
  - dependencies are ok
- The makespan is given by $(N-2)F + m(2F+B) + (N-2)B$ (on GPU_{N-1})

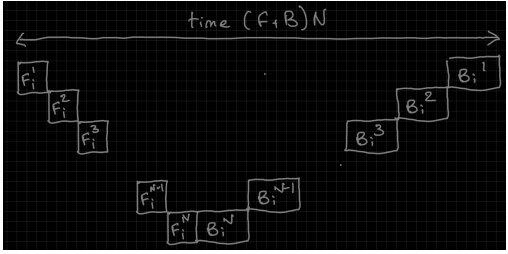
Pipeline parallelism when memory is very small - Lower Bound

- Let us consider GPU_{N-1}. Due to limited memory:
- $\forall i$, without rematerialization, no other F can be performed after the F_i^{N-1} and the end of B_i^{N-1}
- If F_i^{N-1} is kept in memory, the length of the interval dedicated to F_i^{N-1} is
  - at least $F(1) + F(1) + B(2) + B(2)$ if backward cannot be decoupled
  - at least $F(1) + F(1) + B(1) + B(1) + W(1)$ if backward can be decoupled
- which is always larger than $F(1) + B(2)$ or $F(1) + B(1) + W(1)$
- Thus the Critical Path on GPU_{N-1} is at least $(N-2)F + 2m(F+B) + (N-2)B$
- which is higher than what is achieved by previous schedule



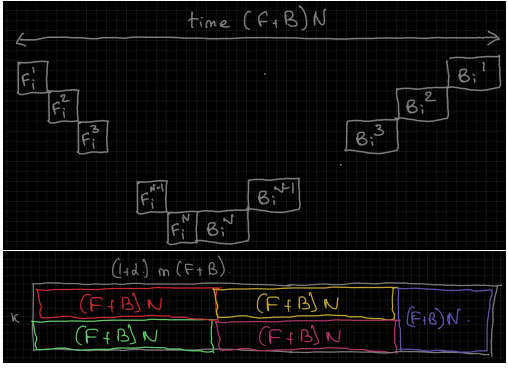
Pipeline parallelism when memory is not large enough - Lower Bound

- Let us denote $K = \lfloor \frac{M_{\text{GPU}}}{M_F} \rfloor$ the number of memory slots
- Let us denote by $\textsc{Makespan} = (1+\alpha)m(F+B)$.
- Let us consider GPU_1
  - $\forall i$, if F_i^1 is never rematerialized until B_i^{N-1}
  - then it must be kept during at least $NF + NB$ time units
  - If m_R F_i^1 are never rematerialized, they will occupy $m_R N(F+B)$ memory slots
  - and there is a total of $K\textsc{Makespan}$ memory slots
- so that $m_R N(F+B) \leq K$ $\textsc{Makespan}$ (memory on GPU_1)
- Moreover, $m(F+B) + (m - m_R)F \leq \textsc{Makespan}$ (work on GPU_1)
- what provides a lower bound for $\textsc{Makespan}$



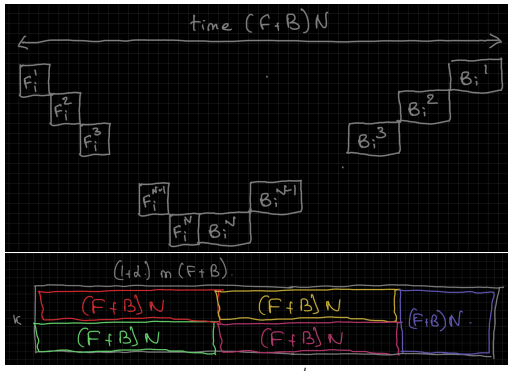
Pipeline parallelism when memory is not large enough - Lower Bound

- Let us denote $K = \lfloor \frac{M_{\text{GPU}}}{M_F} \rfloor$ the number of memory slots
- Let us denote by $\textsc{Makespan} = (1+\alpha)m(F+B)$.
- Let us consider GPU_1
  - $\forall i$, if F_i^1 is never rematerialized until B_i^{N-1}
  - then it must be kept during at least $NF + NB$ time units
  - If m_R F_i^1 are never rematerialized, they will occupy $m_R N(F+B)$ memory slots
  - and there is a total of $K\textsc{Makespan}$ memory slots
- so that $m_R N(F+B) \leq K$ $\textsc{Makespan}$ (memory on GPU_1)
- Moreover, $m(F+B) + (m - m_R)F \leq \textsc{Makespan}$ (work on GPU_1)
- what provides a lower bound for $\textsc{Makespan}$



Pipeline parallelism when memory is not large enough - Lower Bound

- Let us denote $K = \lfloor \frac{M_{\text{GPU}}}{M_F} \rfloor$ the number of memory slots
- Let us denote by $\textsc{Makespan} = (1+\alpha)m(F+B)$.
- Let us consider GPU_1
  - $\forall i$, if F_i^1 is never rematerialized until B_i^{N-1}
  - then it must be kept during at least $NF + NB$ time units
  - If m_R F_i^1 are never rematerialized, they will occupy $m_R N(F+B)$ memory slots
  - and there is a total of $K\textsc{Makespan}$ memory slots
- so that $m_R N(F+B) \leq K$ $\textsc{Makespan}$ (memory on GPU_1)
- Moreover, $m(F+B) + (m-m_R)F \leq \textsc{Makespan}$ (work on GPU_1)
- what provides a lower bound for $\textsc{Makespan}$



$$\alpha_{opt} = \frac{1-\frac{k}{N}}{\frac{k}{N}+\frac{F+B}{F}}$$

Scheduling PP under memory constraint

- Results presented today are mostly negative ones
  - Lower bounds
  - except the agressive rematerialization strategy that achieves (in theory) $\frac{4}{3}$ ratio
- On the positive side, budget is often more than 1 F
  - ILP / Constraint programming can solve rematerialization policies
  - Greedy approach is faster and promising for bigger problems
  - Landscape of scheduling/remat co-optimization is not explored (and not clear)
- A greedy dynamic scheduling approach: very fast and very promising results, but needs to be improved!

Conclusion on Parallelism

GPUs	TP	CP	PP	DP	Seq. Len.	Batch size/DP	Tokens/Batch	TFLOPs/GPU	BF16 MFU
8,192	8	1	16	64	8,192	32	16M	430	43%
16,384	8	1	16	128	8,192	16	16M	400	41%
16,384	8	16	16	8	131,072	16	16M	380	38%

- How to read this ?
- DP data parallelism: rare but expensive computations, *m* scales with DP
- PP pipeline parallelism, *m* scales with PP to have good efficiency (2 PP)
- TP tensor parallelism: parallel matrix computations, a lot of communications, limited in scale, but *m* does not scale with TP
- CP context parallelism or SP sequence parallelism, a lot of communications, limited in scale, but *m* does not scale with CP

  Here 16M tokens / batch is imposed for convergence / accuracy

  So that 16M / 131 072 = 128 is the maximal value for 2 $\mathrm{PP} \times \mathrm{DP}$.

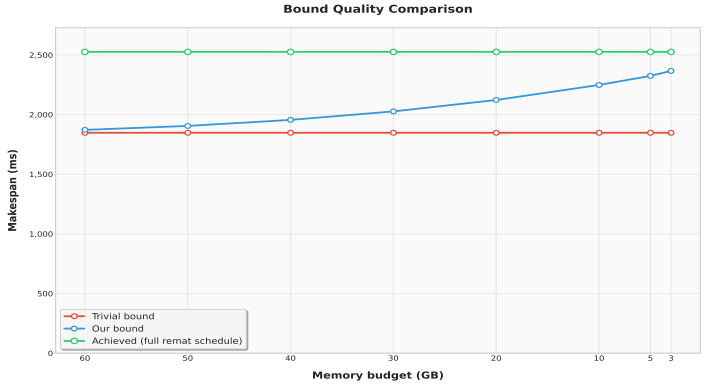
Conclusion

- Training and inference are plenty of nice and difficult scheduling problems
- Difficult Issues:
  - Combination of plenty of different parallelisms
  - Memory issue remains crucial
  - The influence of batch size on convergence / accuracy must be taken into account (plenty of rules of thumb)
  - the stochastic nature of the process opens plenty of opportunities (related to fault tolerance)

Conclusion

- Training and inference are plenty of nice and difficult scheduling problems
- Difficult Issues:
  - Combination of plenty of different parallelisms
  - Memory issue remains crucial
  - The influence of batch size on convergence / accuracy must be taken into account (plenty of rules of thumb)
  - the stochastic nature of the process opens plenty of opportunities (related to fault tolerance)
- But
  - Training these large models really requires a lot of GPUs
  - and for a very long time
  - so that spending time to optimize it is worth the effort
  - and models are quite similar

Bound evaluation



- Trivial bound: computation time $m(F + B)$
- Achieved: schedule presented earlier
- $p = 16$, $m = 32$

How to find α_{opt}

$$m' \geq m(1 - \alpha \frac{F+B}{F})$$

$$m' \leq (1+\alpha)\frac{m}{N}k$$