Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Model Parallelism: Deep Learning con GPU Multiple in modo efficiente e sostenibile

Francesco Gargiulo, Antonio Francesco Gentile, Emilio Greco

RT- ICAR-NA-25-01 **Gennaio 2025**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 8-9C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.icar.cnr.it*
– Sezione di Palermo, Via Ugo La Malfa, 153, 90146 Palermo, URL: *www.icar.cnr.it*

Premessa

Le tecniche descritte in questo lavoro offrono un approccio pratico e completo per affrontare le complessità dell'addestramento distribuito di modelli di deep learning. Grazie a un uso combinato di strategie di parallelismo, ottimizzazioni della memoria e tecnologie avanzate di comunicazione, le soluzioni presentate affrontano le sfide legate alla crescita delle dimensioni dei modelli e alla necessità di maggiori risorse computazionali.

Le tecniche di parallelismo permettono di suddividere il carico di lavoro tra diverse GPU o nodi, riducendo i tempi di calcolo e migliorando l'efficienza. Le ottimizzazioni della memoria, come il checkpointing delle attivazioni o l'uso di precisione mista, aiutano a ridurre il consumo di memoria durante l'addestramento, rendendo possibile l'utilizzo di modelli più grandi anche su hardware con capacità limitate. Infine, le tecnologie di comunicazione, come le librerie per la sincronizzazione dei gradienti, garantiscono che tutte le unità di calcolo lavorino in modo coordinato e coerente.

Le strategie che descriveremo non solo aiutano a superare i limiti fisici della memoria e delle risorse hardware, ma permettono anche di scalare l'addestramento, rendendo possibile il training di modelli complessi su larga scala.

Questo rapporto tecnico fa parte di una collana di documenti tecnici volti a fornire utili informazioni per l'uso efficiente e sostenibile di infrastrutture di calcolo per l'IA basate su architetture multi-GPU. In particolare, questo lavoro è stato reso possibile grazie alle attività realizzate nell'ambito del progetto "Humanities and Cultural Heritage Italian Open Science Cloud – H2IOSC," finanziato dall'Unione europea - NextGenerationEU nell'ambito del PNRR Missione 4, "Istruzione e Ricerca" - Componente 2, "Dalla ricerca all'impresa" - Linea di investimento 3.1, "Fondo per la realizzazione di un sistema integrato di infrastrutture di ricerca e innovazione", decreto di concessione del finanziamento prot. MUR n. 112 del 20-06-2022 (CUP B63C22000730005).

I corsi di formazione forniti da NVIDIA Academy, le competenze interne all'ICAR e la documentazione acquisita attraverso vari canali, hanno permesso di approfondire e applicare tecniche all'avanguardia nel calcolo distribuito consentendo di esplorare e formalizzare modalità operative e casi d'uso delle risorse di calcolo recentemente acquistate dall'ICAR CNR, per il supporto alle ricerche avanzate condotte nell'ambito di infrastrutture per l'AI basate sul paradigma multi-GPU.

Parallelismo dei Dati vs Parallelismo del Modello

Nel lavoro descritto in RT-ICAR-NA-24-04 abbiamo introdotto due approcci fondamentali per l'utilizzo di più GPU nell'addestramento di modelli di deep learning: il Data Parallelism e il Model Parallelism. In particolare, ci siamo focalizzati sull'approfondimento del primo approccio. Per evidenziare i concetti principali, il Data Parallelism consiste nella suddivisione del dataset tra diverse GPU, ciascuna delle quali si occupa di addestrare una copia identica del modello. Durante il processo di addestramento, i gradienti calcolati da ogni GPU vengono sincronizzati, assicurando così che tutte le copie del modello mantengano parametri coerenti e aggiornati.

Questo approccio offre il vantaggio principale di una significativa riduzione dei tempi di addestramento, grazie alla possibilità di scalare facilmente su più GPU, ciascuna delle quali elabora una porzione diversa del dataset. Tuttavia, abbiamo riscontrato alcune limitazioni. Tra queste, un aumento del carico di comunicazione tra le GPU necessario per sincronizzare i gradienti, e la necessità che il modello sia replicabile su ogni GPU, una condizione che può diventare problematica nel caso di modelli di grandi dimensioni. Nel rapporto tecnico, abbiamo illustrato un esempio applicativo che produce risultati positivi, trattando reti neurali per la classificazione delle immagini, dove il dataset può essere suddiviso facilmente senza perdita di informazioni.

Diversamente dal primo approccio, nel Model Parallelism il modello è suddiviso tra più GPU, con ciascuna GPU responsabile di una diversa parte dell'architettura del modello. L'elaborazione dei dati avviene in modo sequenziale tra le GPU. I vantaggi di questo approccio sono principalmente che consente l'addestramento di modelli troppo grandi per essere contenuti nella memoria di una singola GPU come ad esempio modelli di LLM come GPT. Risulta quindi particolarmente adatto per l'addestramento di reti neurali profonde con numerosi strati o modelli e con milioni di parametri. Di contro l'implementazione di questa tecnica risulta più complessa della prima. Anche in questo caso le limitazioni principali dell'approccio consistono nella comunicazione frequente tra le GPU per scambiare attivazioni che può introdurre latenze significative.

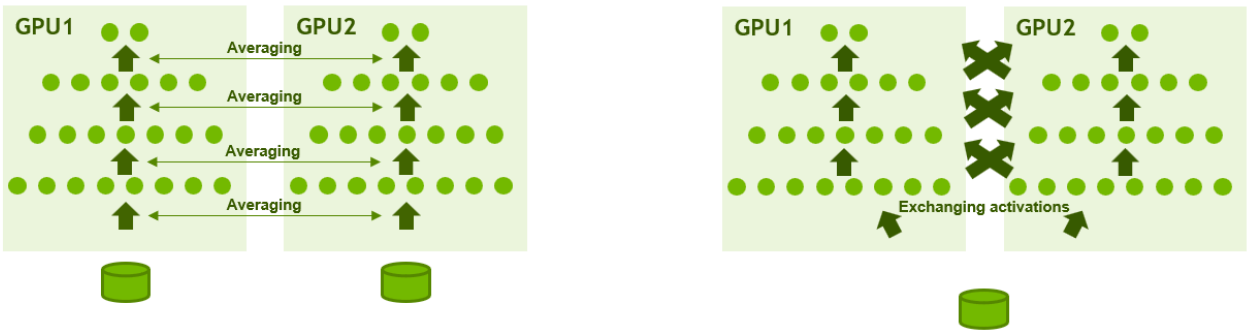
Di seguito una tabella riepilogativa che mette in evidenza le differenze sostanziali tra i due approcci:

Caratteristica	Data Parallelism	Model Parallelism
Dataset	Suddiviso tra le GPU	Lo stesso dataset usato da tutte le GPU

Modello	Identico su ogni GPU	Suddiviso tra le GPU
Comunicazione	Sincronizzazione dei gradienti (peso) tra le GPU	Sincronizzazione delle attivazioni tra le GPU
Utilizzo consigliato	Modelli di dimensioni gestibili e dataset di grandi dimensioni	Modelli molto grandi che superano la memoria di una GPU

Confronto tra Data Parallelism e Model Parallelism

L'immagine riportata di seguito mostra con più chiarezza i diversi tipi di comunicazione che intercorrono tra le GPU in funzione dell'approccio utilizzato:



La parte di comunicazione dunque riveste un ruolo cruciale per la scalabilità e quindi merira un approfondimento particolare.

Prima di procedere con l'analisi in dettaglio dei vari metodi di parallelizzazione del modello conosciuti allo stato dell'arte, ci soffermeremo ancora a descrivere alcuni concetti ad alto livello. In RT-ICAR-NA-24-04 abbiamo visto un esempio paratico di trasformazione di una attività di addestramento su singola GPU, la sequenza operativa delle attività di addestramento su una singola GPU comprende:

1. **Lettura dei dati**: Il dataset viene prelevato dal disco e trasferito alla GPU.
2. **Pre-elaborazione dei dati**: Trasformazioni come normalizzazione o incremento dei dati.
3. **Forward Propagation**: Calcolo delle attivazioni e della previsione per ogni strato della rete neurale.
4. **Calcolo della funzione di perdita**: Valutazione dell'errore rispetto al valore atteso.

5. **Backward Propagation**: Calcolo dei gradienti attraverso ogni strato (retropropagazione).
6. **Aggiornamento dei pesi**: Applicazione dei gradienti per aggiornare i parametri del modello.
7. **Restituzione del controllo**: Il ciclo si ripete fino al completamento delle epoche.

Abbiamo visto che per adattare l'attività per distribuirla su più GPU occorre aggiungere ulteriori fasi descritte di seguito:

- **Suddivisione del dataset o del modello**, in base all'approccio scelto (Data Parallelism o Model Parallelism)
- **Sincronizzazione tra le GPU,** che a questo punto diventa essenziale, sia per i gradienti (Data Parallelism) che per le attivazioni (Model Parallelism).
- **Ottimizzare della comunicazione tra le GPU** per evitare colli di bottiglia che ridurrebbero l'efficacia del parallelismo. Tecniche come l'uso della libreria **NVIDIA NCCL** possono ridurre significativamente le latenze per il trasferimento dei dati tra GPU.

Sta quindi allo sviluppatore bilanciare adeguatamente, in questa trasformazione, il carico tra le GPU per evitare sprechi di risorse e garantire prestazioni ottimali. Nel capitolo successivo mostreremo gli strumenti messi a disposizione dalla libreria DDP per affrontare queste sfide.

Tecniche di sincronizzazione ed ottimizzazione della comunicazione tra le GPU

In RT-ICAR-NA-24-04 abbiamo già descritto l'importanza dell'operazione **All-reduce** in merito alla fase di sincronizzazione tra le GPU. All-reduce è un'operazione cruciale nell'addestramento distribuito, in quanto garantisce la sincronizzazione degli aggiornamenti tra le GPU. Sebbene introduca un certo sovraccarico, NVIDIA ha ottimizzato i progressi nell'hardware e nelle librerie rendendo questa operazione un metodo efficiente per supportare carichi di lavoro di deep learning su larga scala.

L'operazione consente di combinare i dati provenienti da diverse GPU, applicare una funzione di riduzione (come somma, minimo o massimo) e distribuire il risultato a tutte le GPU partecipanti. Le fasi dell'operatore, nel caso di Data Parallelism si possono quindi sintetizzare in:

1. **Calcolo Locale dei Gradienti:** Ogni GPU calcola i gradienti basati sul proprio sottoinsieme di dati.

2. **Riduzione dei Gradienti:** I gradienti calcolati vengono combinati attraverso un'operazione di riduzione (ad esempio, una somma) per ottenere un risultato aggregato.
3. **Distribuzione del Risultato:** Il risultato della riduzione viene inviato a tutte le GPU, assicurando che ciascuna disponga degli stessi parametri aggiornati.

Questo processo garantisce che tutte le GPU siano sincronizzate e che l'addestramento proceda in modo coerente su tutte le unità. Per fare un esempio pratico, consideriamo un'operazione di somma All-Reduce tra 4 GPU, ciascuna con un array di valori:

- **GPU 0:** [1, 2, 3]
- **GPU 1:** [4, 5, 6]
- **GPU 2:** [7, 8, 9]
- **GPU 3:** [10, 11, 12]

Dopo l'operazione di All-Reduce (somma), ogni GPU riceverà lo stesso array risultante dalla somma elemento per elemento: **Risultato:** [22, 26, 30]

Questo significa che ogni elemento dell'array risultante è la somma dei corrispondenti elementi degli array originali di tutte le GPU. Se anche solo uno dei rank non partecipa all'operazione, gli altri rank rimarranno in attesa che il processo mancante si unisca. Questo può causare un **blocco indefinito** (deadlock) del sistema, perché le operazioni collettive non possono essere completate senza la partecipazione di tutti i rank previsti.

Oltre al AllReduce la libreria NCCL (NVIDIA Collective Communication Library) offre metodi fondamentali per gestire la comunicazione e la sincronizzazione tra GPU in scenari di addestramento distribuito che prevedono anche metodi ottimizzati per il Model Parallelism.

NCCL consente di:

- Aggregare e sincronizzare i gradienti (**AllReduce**)
- Condividere parametri iniziali (**Broadcast**)
- Suddividere i carichi di lavoro o raccogliere risultati (**ReduceScatter e Reduce**)
- Integrare e distribuire informazioni globali (**AllGather**)

La **Broadcast Operation** prevede che una GPU, spesso definita come nodo principale o "master", possegga un **dato iniziale come i parametri del modello** e lo distribuisca a tutte le altre GPU partecipanti. Questa operazione è unidirezionale: i dati fluiscono dalla GPU sorgente verso le altre GPU senza alcuna aggregazione o elaborazione aggiuntiva.

La **Reduce Operation** combina i dati da tutte le GPU applicando un'operazione di riduzione (ad esempio, somma o massimo) e invia il risultato aggregato a una singola GPU (destinazione). Viene utilizzato per il calcolo del gradiente totale su una GPU principale per aggiornare i parametri del modello.

La **AllGather Operation** raccoglie i dati locali da ogni GPU e li distribuisce a tutte le altre GPU partecipanti. Questo metodo risulta particolarmente utile per sincronizzare o condividere informazioni. Un esempio d'uso è l'aggregazione di risultati intermedi durante il calcolo distribuito.

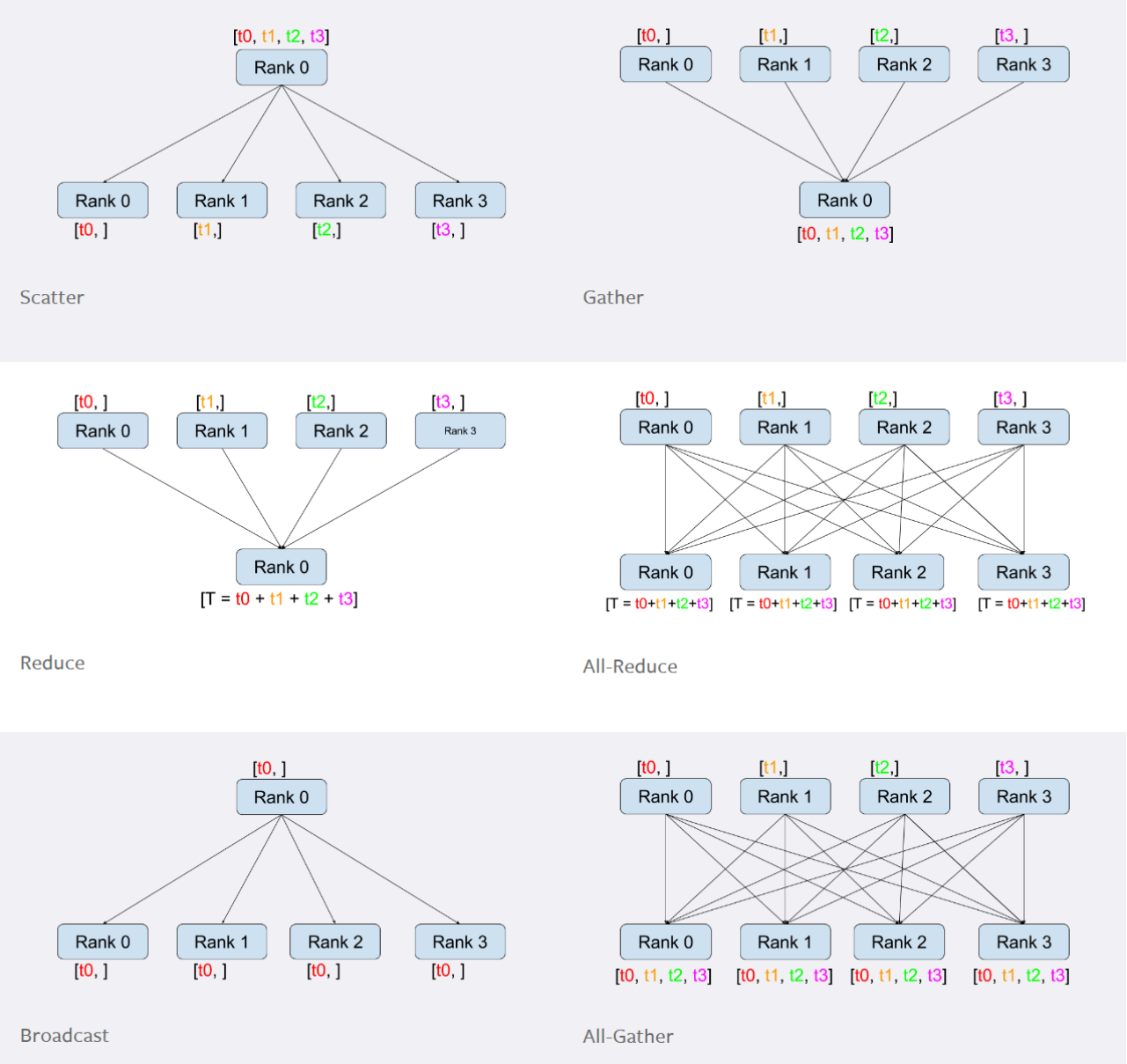
La **ReduceScatter Operation** combina i dati da tutte le GPU applicando un'operazione di riduzione (come somma o massimo) e distribuisce le diverse parti del risultato aggregato tra le GPU. Un esempio d'uso di questa operazione è per suddividere il carico computazionale tra le GPU durante l'addestramento distribuito.

Di seguito una tabella riepilogativa dei metodi di comunicazione/sincronizzazione delle GPU:

Operazione	Descrizione Principale	Scopo
AllReduce	Aggrega e distribuisce i dati a tutte le GPU	Sincronizzazione globale dei parametri
Broadcast	Copia i dati da una GPU a tutte le altre	Inizializzazione condivisa
Reduce	Aggrega i dati su una GPU specifica	Raccolta centralizzata
AllGather	Raccoglie dati da tutte le GPU e li distribuisce	Condivisione di informazioni globali
ReduceScatter	Combina i dati e distribuisce porzioni tra le GPU	Suddivisione del carico computazionale

Oltre ai metodi descritti in precedenza, la libreria **NCCL** include tecniche di ottimizzazione altamente sofisticate che migliorano l'efficienza e la scalabilità delle operazioni collettive in ambienti distribuiti. Tra queste, l'**Algoritmo Ring All-Reduce** rappresenta una soluzione particolarmente efficace, basata su una tecnica di comunicazione con topologia ad anello che permette alle GPU di scambiare dati solo con i vicini. Questo approccio riduce significativamente il traffico di rete globale e garantisce una scalabilità lineare con il numero di GPU, pur risentendo di una latenza più elevata nelle configurazioni multi-nodo. Un'altra strategia è l'**All-Reduce**

Gerarchico, che combina riduzioni locali all'interno dei nodi con riduzioni globali tra nodi diversi. Questo metodo sfrutta tecnologie come NVLink per la comunicazione intra-nodo e reti ad alta velocità come InfiniBand per la comunicazione inter-nodo, risultando ideale per configurazioni distribuite su larga scala, sebbene richieda infrastrutture hardware avanzate.



Un'ulteriore ottimizzazione è rappresentata dalle **tecniche di compressione**, che riducono il volume di dati trasferiti tra le GPU comprimendo i gradienti o **rappresentandoli con precisione ridotta**. Tecniche come la quantizzazione, la sparsificazione e la compressione dei dati consentono di diminuire il carico di rete e accelerare le operazioni collettive, mantenendo comunque prestazioni computazionali elevate. Tuttavia, queste strategie possono introdurre una perdita di precisione nei

gradienti, influenzando la convergenza del modello, e aumentare il carico computazionale per la compressione e la decompressione.

Questi metodi ottimizzati, integrati nella libreria NCCL, permettono di affrontare le sfide della sincronizzazione durante l'addestramento distribuito, garantendo che anche i modelli più complessi possano essere scalati efficientemente su infrastrutture multi-GPU e multi-nodo.

Tecniche di Ottimizzazione della Memoria

La tecnica di Model Parallelism risulta indispensabile quando un modello è troppo grande per risiedere su una sola GPU. Quando si lavora con modelli di scala miliardaria, il consumo di memoria rappresenta uno dei maggiori ostacoli. Questo è dovuto alla necessità di memorizzare diversi componenti fondamentali: i parametri del modello, i gradienti calcolati durante il backpropagation e gli stati dell'ottimizzatore come il momentum e le varianze richieste da algoritmi **come Adam**. Tuttavia, il consumo di memoria totale non deriva solo da questi componenti, ma anche da altri fattori, come le attivazioni memorizzate durante il forward pass, i buffer temporanei utilizzati nelle operazioni intermedie e la memoria frammentata, che non può essere sfruttata efficacemente.

Per affrontare queste problematiche, sono state sviluppate diverse strategie avanzate. Una di queste è l'uso della precisione mista nota come **Mixed Precision Training**[1], che consente di memorizzare i parametri del modello in due formati differenti:

- FP16, per eseguire calcoli più efficienti e ridurre l'occupazione di memoria
- FP32, per garantire stabilità numerica

In questo approccio, i gradienti vengono memorizzati in FP16 per risparmiare memoria, mentre gli stati dell'ottimizzatore, che richiedono maggiore precisione, rimangono in FP32. Questo compromesso permette di ottimizzare il consumo di memoria senza compromettere la qualità dell'addestramento.

[1] *Mixed precision training*. International Conference on Learning Representations (ICLR). Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., & Wu, H. (2018). Disponibile al link: https://arxiv.org/abs/1710.03740

Un altro approccio significativo è rappresentato dalla tecnica **ZeRO-Offload**, un'estensione dell'ottimizzatore **ZeRO**[2], che **redistribuisce parte del carico di memoria dalla GPU alla CPU**. In particolare, stati del modello come gradienti e stati dell'ottimizzatore vengono spostati nella memoria della CPU, liberando spazio prezioso sulla GPU per gestire attivazioni e operazioni critiche. ZeRO-Offload non solo consente di addestrare modelli di grandi dimensioni anche su hardware con memoria limitata, ma ottimizza anche l'uso della CPU per gestire calcoli intensivi come gli aggiornamenti dei parametri, grazie a implementazioni ottimizzate di algoritmi come Adam. Inoltre, la tecnica utilizza un'aggiornamento ritardato dei parametri (Delayed Parameter Update), che permette di sovrapporre il calcolo sulla GPU con gli aggiornamenti sulla CPU, mascherando i ritardi di comunicazione e migliorando l'efficienza del throughput.

Un altro fattore critico nella gestione della memoria è rappresentato dalle **attivazioni, che consumano una quantità significativa di memoria durante l'addestramento**. Per affrontare questa sfida, sono state sviluppate tecniche avanzate come **Activation Checkpointing**[3] e **Selective Activation Recomputation**, entrambe evoluzioni del **Gradient Checkpointing**[4] tradizionale. Activation Checkpointing suddivide i calcoli in segmenti, salvando selettivamente solo un sottoinsieme di attivazioni e ricalcolando le restanti durante il backpropagation, consentendo così un notevole risparmio di memoria senza sacrificare la precisione del modello. Selective Activation Recomputation, invece, si concentra esclusivamente sulle attivazioni che offrono il massimo beneficio in termini di risparmio di memoria, mantenendo al minimo il sovraccarico computazionale. **Questi approcci combinati possono consentire un risparmio di memoria fino al 70%**, con un impatto minimo sul tempo di calcolo, riducendo ulteriormente il carico sulle GPU.

Parallelismo del Modello

Il Model Parallelism rappresenta un approccio essenziale per l'addestramento di modelli di deep learning di grandi dimensioni, poiché permette di suddividere il carico computazionale tra più GPU. Questa strategia consente di superare i limiti di memoria e di scalare a modelli con miliardi di

[2] ZeRO (Zero Redundancy Optimizer) è una tecnica avanzata per ottimizzare l'addestramento distribuito di modelli di deep learning di grandi dimensioni, riducendo la ridondanza nella memorizzazione dei parametri, gradienti e stati dell'ottimizzatore tra le GPU. Per maggiori dettagli, si veda: Rajbhandari, S., Rasley, J., Ruwase, O., & He, Y. (2020). *ZeRO: Memory Optimization Towards Training A Trillion Parameter Models*. Disponibile su: https://arxiv.org/abs/1910.02054.

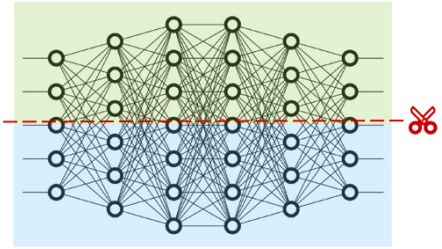
[3] Training Deep Nets with Sublinear Memory Cost. *Timothy P. Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin.* **Conference:** Advances in Neural Information Processing Systems (*NeurIPS*), 2016. Link: https://arxiv.org/abs/1604.06174

[4] Gradient Checkpointing è una tecnica per ridurre il consumo di memoria durante l'addestramento di modelli di deep learning, ricalcolando alcune attivazioni durante il backpropagation anziché memorizzarle. Per maggiori dettagli, si veda: Chen, T., Xu, B., Zhang, C., & Guestrin, C. (2016). *Training Deep Nets with Sublinear Memory Cost*. Disponibile su: https://arxiv.org/abs/1604.06174.

parametri, utilizzando tecniche avanzate come il **Tensor Parallelism**, il **Pipeline Parallelism** e il **Sequence Parallelism**.

Tensor Parallelism

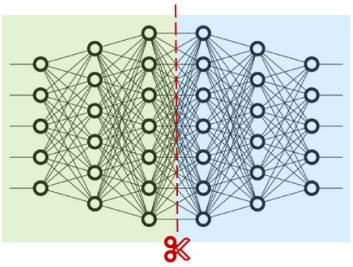
Il Tensor Parallelism suddivide i calcoli di ciascun layer del modello tra più GPU. Questo significa che ogni layer è "scomposto" e distribuito tra le GPU, che lavorano contemporaneamente per elaborare diverse parti dei calcoli associati al layer. Ad esempio, nel caso di modelli Transformer, gli strati di attenzione o feed-forward possono essere suddivisi tra GPU, consentendo di ridurre i requisiti di memoria per singola GPU. Questo approccio è particolarmente efficace all'interno di un singolo nodo, come in un server DGX A100, dove le GPU sono collegate tramite NVLink, un'interconnessione ad alta larghezza di banda. NVLink riduce significativamente i costi di comunicazione, migliorando le prestazioni complessive. Una delle principali difficoltà del Tensor Parallelism è **la necessità di un'alta larghezza di banda per sincronizzare i dati tra le GPU**, poiché le operazioni All-Reduce necessarie per combinare i risultati intermedi possono diventare un collo di bottiglia in configurazioni distribuite su più nodi.



Il Tensor Parallelism suddivide i singoli strati della rete neurale tra più GPU. Questo significa che ciascun layer è elaborato simultaneamente da più GPU, ognuna delle quali gestisce una parte del calcolo complessivo del layer. Con questa tecnica, tutte le GPU possono lavorare contemporaneamente, poiché non c'è una dipendenza sequenziale tra le operazioni. Tuttavia, questo approccio richiede una **comunicazione aggiuntiva tra le GPU per sincronizzare i risultati** dei calcoli, soprattutto nelle operazioni di All-Reduce, che possono essere costose.

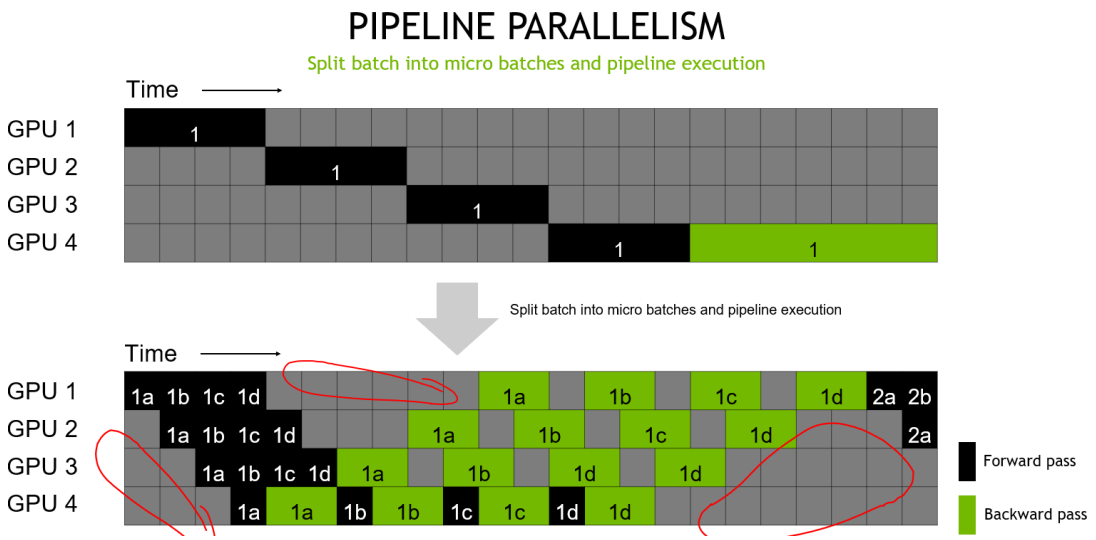
Pipeline Parallelism

Il Pipeline Parallelism, invece, distribuisce gruppi di strati consecutivi del modello su GPU diverse. Ad esempio, in un modello a sei strati, i primi tre strati potrebbero essere assegnati a una GPU e i successivi tre a un'altra. Durante l'addestramento, **i dati attraversano questa pipeline**, passando da una GPU alla successiva. A differenza del Tensor Parallelism, il Pipeline Parallelism utilizza una comunicazione **point-to-point** tra le GPU, che **è meno costosa e può essere facilmente implementata anche in ambienti multi-nodo**. Uno svantaggio significativo è rappresentato dalle **"bubbles" di inattività**, che si verificano quando alcune GPU restano inutilizzate in attesa dei dati da altre GPU. Questo problema è particolarmente evidente con batch piccoli. Per ridurre l'impatto delle bubbles, si utilizza spesso il concetto di micro-batch, che consente di parallelizzare ulteriormente il lavoro all'interno della pipeline.



Nell'esempio illustrato nella parte superiore, una rete composta da sei strati viene distribuita su due GPU, rappresentate rispettivamente in verde e blu. In questo schema, ogni GPU è responsabile di un gruppo di tre strati consecutivi, e la comunicazione avviene solo nei punti in cui gli strati sono suddivisi tra le GPU.

Mentre nell'immagine successiva si evidenziano i limiti del Pipeline Parallelism. Da un lato la tecnica consente di dividere il modello tra più GPU per gestire carichi computazionali maggiori, dall'altro, la sequenzialità del processo introduce un limite alla scalabilità. Ciò ne consegue che **l'aggiunta di più GPU può non tradursi in un incremento lineare della velocità** a causa delle dipendenze sequenziali. L'uso di ottimizzazioni come **micro-batch e overlapping dei calcoli** può mitigare alcune di queste limitazioni, migliorando la scalabilità e l'efficienza complessiva del sistema.



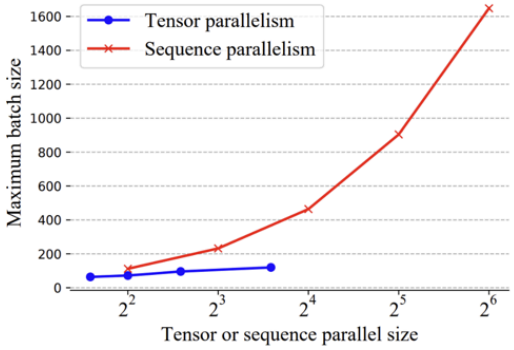
Sequence Parallelism

Il Sequence Parallelism rappresenta una strategia avanzata per ottimizzare l'addestramento distribuito di modelli che elaborano sequenze lunghe, come quelli basati su Transformer[5], spesso utilizzati nell'elaborazione del linguaggio naturale (NLP). L'idea alla base di questo approccio è quella di suddividere le sequenze di dati tra le GPU, assegnando a ciascuna GPU una porzione della sequenza da elaborare. A differenza del Data Parallelism, che divide il dataset in batch indipendenti, il Sequence Parallelism lavora su un'unica sequenza, permettendo di gestire sequenze più lunghe senza eccedere i limiti di memoria delle singole GPU.

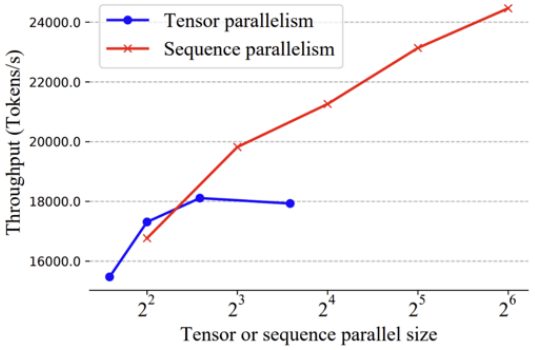
Durante l'addestramento, i parametri del modello vengono condivisi tra tutte le GPU coinvolte, garantendo che ognuna abbia accesso alle stesse informazioni. Ogni GPU elabora la sua porzione della sequenza, generando risultati intermedi che vengono poi distribuiti alle altre GPU per completare i calcoli necessari. Questo schema consente un'elevata efficienza computazionale e permette di addestrare modelli che altrimenti non potrebbero essere gestiti a causa delle limitazioni di memoria.

Tuttavia, il Sequence Parallelism presenta alcuni limiti, soprattutto in termini di comunicazione. Con GPU distribuite su più nodi, i costi di comunicazione aumentano significativamente, poiché i risultati intermedi devono essere trasferiti frequentemente tra i nodi. Questo problema è particolarmente rilevante per modelli di grandi dimensioni come i Transformer che richiedono un'elevata quantità di operazioni di sincronizzazione e trasferimento dati tra GPU.

[5] *Transformer* è un'architettura di rete neurale introdotta da Vaswani et al. **"Attention is All You Need"** (2017). È alla base di modelli avanzati come BERT, GPT e T5. Disponibile online: https://arxiv.org/abs/1706.03762.



(a) Maximum batch size of BERT Base scaling along tensor or sequence parallel size



(b) Throughput of BERT Base scaling along tensor or sequence parallel size

La figura tratta dall'articolo "Sequence Parallelism: Long Sequence Training from System Perspective" di Shenggui Li et al. (2022) illustra le differenze tra il Tensor Parallelism e il Sequence Parallelism nell'addestramento del modello BERT.

Nel caso di Tensor Parallelism, ogni layer del modello BERT è suddiviso tra più GPU. Ogni GPU elabora una porzione del calcolo del layer, e i risultati intermedi vengono combinati attraverso operazioni di comunicazione collettiva, come l'All-Reduce. Questo approccio consente di distribuire il carico computazionale di un singolo layer tra più GPU, ma richiede una comunicazione intensiva per sincronizzare i risultati intermedi. Nell'approccio Sequence Parallelism, la sequenza di input viene suddivisa in segmenti, ciascuno assegnato a una GPU diversa. Ogni GPU elabora la propria porzione della sequenza attraverso tutti i layer del modello. Per calcolare correttamente l'attenzione, le embedding delle chiavi e dei valori vengono scambiate tra le GPU in una struttura ad anello, implementando la **Ring Self-Attention (RSA)**. Questo metodo consente di gestire sequenze più lunghe distribuendo il carico di memoria e calcolo tra le GPU, riducendo la necessità di comunicazione collettiva intensiva.

Combinazione delle Tecniche

Il Pipeline Parallelism viene spesso utilizzato per suddividere i checkpoint e creare **ensemble di modelli**, consentendo di addestrare reti molto profonde dividendo il carico di lavoro. Al contrario, il Tensor Parallelism è comunemente implementato quando si lavora con un **singolo modello di grandi dimensioni** che viene suddiviso a livello di framework tra GPU. Questo approccio consente di condividere layer ed embedding tra le GPU, rendendo possibile l'addestramento simultaneo. Qundo usare uno o l'altro dipende dal contesto, di seguito alcuni scenari:

Tensor Parallelism intra-nodo: Funziona al meglio all'interno di un singolo nodo, come un server DGX A100, dove le GPU sono collegate tramite NVLink, che offre un'elevata larghezza di banda. Tuttavia, l'operazione di all-reduce richiesta dal Tensor Parallelism può risultare costosa anche in questo scenario.

Pipeline Parallelism inter-nodo: È più adatto per configurazioni multi-nodo, dove gli strati del modello sono distribuiti tra server DGX A100. Questo approccio utilizza comunicazioni point-to-point, che sono più economiche e possono essere eseguite tra nodi senza creare colli di bottiglia significativi.

Un aspetto fondamentale evidenziato nella documentazione NVIDIA e negli esperimenti presentati è che la combinazione di Tensor Parallelism e Pipeline Parallelism, insieme al Data Parallelism, consente di scalare i modelli a dimensioni enormi, come nel caso di modelli con un trilione di parametri. Ad esempio, nel training di un modello **GPT-3** con 175 miliardi di parametri, è stato possibile ottenere un throughput migliorato di 114 volte, passando da un modello con 1 miliardo di parametri su 32 GPU a un modello con 1 trilione di parametri su 3072 GPU A100[6]. L'uso combinato di Tensor Parallelism e Pipeline Parallelism ha permesso di completare l'addestramento del modello GPT-3 in poco più di un mese su 1024 GPU del supercomputer NVIDIA Selene[7].

Un esempio pratico di modelli scalabili è **Megatron-LM[8]**, dove il Sequence Parallelism viene combinato con altre tecniche di parallelismo, come il Tensor Parallelism e il Pipeline Parallelism, per bilanciare il carico computazionale e minimizzare i costi di comunicazione. Ad esempio, nei modelli Transformer, i layer di attenzione possono beneficiare del Sequence Parallelism, poiché le GPU possono collaborare per elaborare input di sequenze molto lunghe, mantenendo alta l'efficienza e riducendo la necessità di frammentare il modello o il dataset.

Un altro esempio di tecniche avanzate di parallelizzazione per ottimizzare l'addestramento e **NVIDIA NeMo[9] Framework.** Una piattaforma cloud-native progettata per facilitare la creazione, la personalizzazione e la distribuzione di modelli di intelligenza artificiale generativa su larga scala. NeMo ottimizza l'utilizzo delle risorse hardware, consentendo l'addestramento di modelli con miliardi o trilioni di parametri, attraverso l'integrazione di Tensor Parallelism e Pipeline Parallelism.

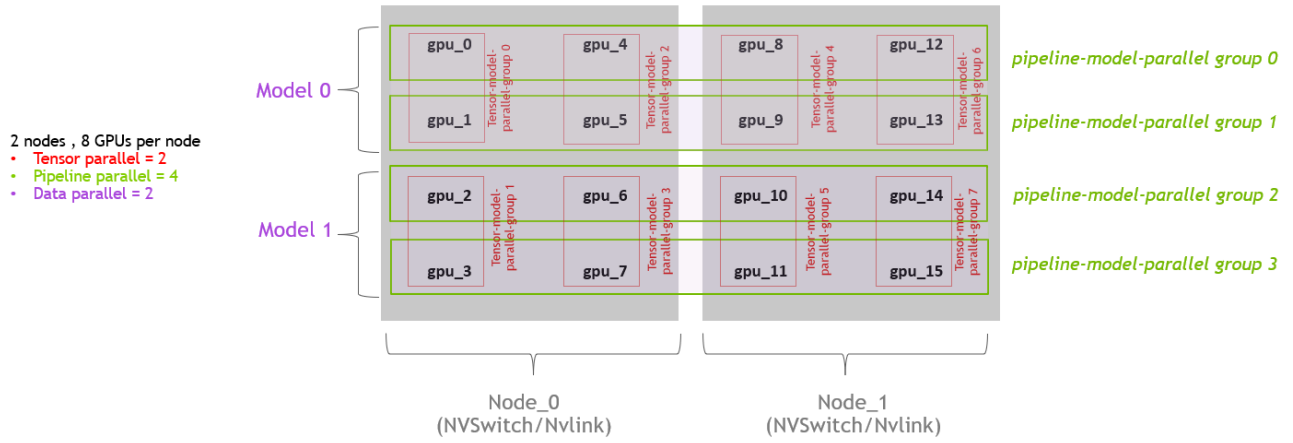
---

[6] https://developer.nvidia.com/blog/scaling-language-model-training-to-a-trillion-parameters-using-megatron/

[7] https://resources.nvidia.com/en-us-dgx-systems/making-selene?xs=490164

[8] *Megatron* è un'infrastruttura sviluppata da NVIDIA per l'addestramento di modelli Transformer di grandi dimensioni, ottimizzata per sfruttare GPU moderne e tecniche avanzate di parallelismo come il Tensor Parallelism e il Pipeline Parallelism. Permette di scalare modelli fino a trilioni di parametri, rendendo possibile l'addestramento di modelli come GPT-3 in ambienti distribuiti. https://github.com/NVIDIA/Megatron-LM

[9] https://docs.nvidia.com/nemo-framework/



L'immagine illustra una configurazione avanzata del framework NVIDIA NeMo, utilizzata per l'addestramento distribuito di modelli di deep learning su un'infrastruttura composta da due nodi, ciascuno con otto GPU. Questo approccio si basa sull'integrazione di più tecniche di parallelismo per massimizzare l'efficienza computazionale e la scalabilità, affrontando in modo efficace le sfide poste dai modelli di grandi dimensioni. All'interno di ogni nodo, le GPU sono connesse tramite NVLink, una tecnologia che fornisce un'elevata larghezza di banda per la comunicazione interna, riducendo i tempi di trasferimento dei dati tra GPU. I due nodi, a loro volta, sono interconnessi per consentire il coordinamento delle operazioni distribuite su tutte le sedici GPU disponibili. La configurazione combina il Tensor Parallelism, il Pipeline Parallelism e il Data Parallelism, ciascuno con un ruolo specifico nel processo di addestramento.

Il Tensor Parallelism si concentra sulla suddivisione dei calcoli all'interno di ciascun layer del modello tra più GPU. In questo caso, le GPU di un singolo nodo lavorano in modo coordinato per elaborare i calcoli di un layer specifico, sfruttando la comunicazione ad alta velocità fornita da NVLink. Questo permette di ridurre il carico di memoria su ogni GPU, distribuendo il lavoro in modo equilibrato.

Il Pipeline Parallelism, invece, suddivide il modello in gruppi di strati consecutivi, assegnando ciascun gruppo a diverse GPU. In questa configurazione, le operazioni sono organizzate in una pipeline, dove i dati passano da una GPU all'altra in modo sequenziale, consentendo a ciascuna GPU di lavorare su una parte distinta del modello. Sebbene questo approccio introduca una certa dipendenza temporale tra le operazioni, è bilanciato dall'efficienza garantita dalla comunicazione point-to-point, che risulta meno onerosa rispetto a operazioni collettive come l'all-reduce.

Il Data Parallelism completa il sistema replicando il modello su entrambi i nodi e suddividendo il dataset tra di essi. Ogni nodo elabora una porzione distinta dei dati e i gradienti calcolati vengono

sincronizzati tra i nodi, garantendo che i pesi del modello rimangano consistenti. Questa sincronizzazione avviene utilizzando tecniche efficienti per minimizzare i costi di comunicazione.
In questa configurazione, NVIDIA NeMo dimostra come l'integrazione di più tecniche di parallelismo possa consentire l'addestramento di modelli con miliardi di parametri su infrastrutture distribuite. Il Tensor Parallelism è particolarmente efficace all'interno di un singolo nodo grazie alla velocità di NVLink, mentre il Pipeline Parallelism e il Data Parallelism si combinano per scalare il sistema su più nodi, gestendo sia la comunicazione intra-nodo che inter-nodo. Questo approccio non solo ottimizza l'utilizzo delle risorse hardware, ma rende possibile l'addestramento di modelli di deep learning su larga scala in tempi ridotti e con un'efficienza elevata.

Mixture of Experts (MoE)

Il concetto di Mixture of Experts (MoE) rappresenta un'innovazione significativa nel campo del deep learning, fornendo un approccio efficiente e scalabile per l'addestramento e l'inferenza di modelli di grandi dimensioni. La filosofia alla base di MoE si basa sull'idea di suddividere il modello in moduli indipendenti, detti "esperti", ciascuno specializzato in un sottoinsieme del compito complessivo. Questo approccio non solo migliora l'efficienza computazionale, ma consente anche di integrare e ottimizzare tecniche avanzate di calcolo parallelo.
Il Mixture of Experts non è un concetto nuovo. La sua prima formulazione risale al 1991, quando *Jacobs et al*. proposero un sistema composto da più esperti locali, ciascuno responsabile di una specifica area del problema, e una rete di gating responsabile di determinare quale esperto utilizzare. Questo disaccoppiamento dei pesi riduceva le interferenze tra i diversi moduli, accelerando l'apprendimento e migliorando la generalizzazione. Tuttavia, la vera rivoluzione di MoE è arrivata con i recenti sviluppi nel campo del deep learning, che hanno permesso di applicare questo concetto a modelli su larga scala.

L'evoluzione di Mixture of Experts

Con il lavoro di *Noam Shazeer et al*. del 2017, MoE è stato applicato a modelli di linguaggio di grandi dimensioni basati su reti neurali ricorrenti (RNN). Questa implementazione sfruttava una sparsità strategica, in cui solo un sottoinsieme degli esperti veniva attivato per ogni input. Questo approccio riduceva significativamente i costi computazionali senza sacrificare l'accuratezza del modello. Più recentemente, i **Switch Transformers**, proposti da *William Fedus et al.*, hanno ulteriormente sviluppato questo concetto, implementando MoE nei modelli Transformer. Attraverso

una selezione efficiente degli esperti, questi modelli sono stati in grado di scalare fino a **trilioni di parametri**, mantenendo un'efficienza computazionale elevata grazie all'attivazione sparsa.
Un ulteriore passo avanti è rappresentato dal lavoro più recente, **Mixtral of Experts**, che introduce miglioramenti specifici per l'integrazione di MoE nei modelli Transformer. In questo approccio, i layer feed-forward vengono sostituiti con layer basati su MoE, e ogni token influenza un sottoinsieme degli esperti, ottimizzando ulteriormente l'uso delle risorse computazionali.

Integrazione nel Calcolo Parallelo

Il Mixture of Experts si inserisce naturalmente nel contesto del calcolo parallelo, sfruttando e ottimizzando tecniche come il Model Parallelism, il Data Parallelism, e il Pipeline Parallelism. La struttura modulare di MoE consente di distribuire gli esperti su diverse GPU o nodi, sfruttando il Model Parallelism per dividere il carico computazionale. Allo stesso tempo, il gating network utilizza il Data Parallelism per distribuire i dati di input tra le GPU, mentre il Pipeline Parallelism può essere utilizzato per gestire il flusso di dati tra gli strati del modello.
Una delle caratteristiche più distintive di MoE è la sua **sparsità intrinseca**, che consente di attivare solo un sottoinsieme di esperti per ogni input. Questo riduce il numero di calcoli necessari, migliorando l'efficienza e riducendo il carico di memoria. Inoltre, grazie alla sparsità, il MoE si combina efficacemente con tecniche di ottimizzazione come **ZeRO-Offload**, che sposta stati come gradienti e parametri sulla memoria della CPU, liberando spazio sulla GPU per le operazioni critiche.
Il **Sequence Parallelism** completa questa strategia, suddividendo la sequenza di input tra le GPU. In questo contesto, MoE riduce ulteriormente i costi computazionali e di memoria, poiché ogni GPU deve elaborare solo un sottoinsieme degli esperti per la propria parte della sequenza. Inoltre, la comunicazione tra GPU è ottimizzata attraverso la **Ring Self-Attention**, riducendo il sovraccarico rispetto a tecniche come l'All-Reduce.

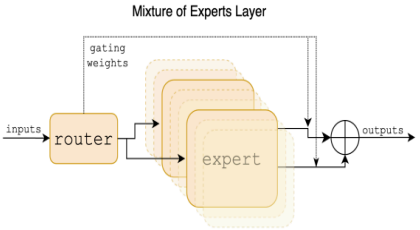


Figure 1: Mixture of Experts Layer. Each input vector is assigned to 2 of the 8 experts by a router. The layer's output is the weighted sum of the outputs of the two selected experts. In Mixtral, an expert is a standard feedforward block as in a vanilla transformer architecture.

IO-Aware Attention Algorithm e FlashAttention nel Model Parallelism

L'IO-Aware Attention Algorithm, con particolare riferimento alla sua implementazione in FlashAttention proposta da *Dao et al*. nel 2022, rappresenta un progresso significativo nel campo del deep learning, affrontando in modo innovativo le problematiche legate all'efficienza computazionale e alla gestione della memoria. Questo algoritmo è stato sviluppato per ottimizzare i calcoli dell'attenzione, una delle operazioni più costose e centrali nei modelli Transformer, introducendo miglioramenti che lo rendono altamente compatibile con le tecniche di Model Parallelism.

FlashAttention è stato progettato per ridurre il costo quadratico associato ai calcoli dell'attenzione rispetto alla lunghezza della sequenza di input, che tradizionalmente rappresenta un ostacolo per l'efficienza dei Transformer. L'algoritmo adotta un approccio orientato alla gestione delle operazioni di input/output (IO), minimizzando i costi legati agli accessi alla memoria. In particolare, FlashAttention si basa su **una tecnica chiamata tiling**, che suddivide i dati in blocchi gestibili e sfrutta al massimo la memoria veloce on-chip, come la SRAM, riducendo il numero di accessi alla memoria off-chip, più lenta. Questo approccio permette di combinare il calcolo e i trasferimenti di memoria in un unico processo ottimizzato, migliorando notevolmente il throughput computazionale.

Una delle principali innovazioni di **FlashAttention risiede nella sua capacità di scalare linearmente rispetto alla lunghezza della sequenza**. A differenza dei metodi tradizionali, che presentano un costo computazionale quadratico, FlashAttention gestisce in modo efficiente sequenze molto lunghe senza aumentare in modo significativo il carico computazionale o il consumo di memoria. Questo lo rende particolarmente utile in applicazioni come il processamento di sequenze lunghe in modelli di linguaggio naturale o in altre architetture Transformer su larga scala.

In termini di integrazione con il Model Parallelism, FlashAttention si dimostra altamente compatibile. Grazie alla sua natura ottimizzata, l'algoritmo distribuisce il carico computazionale dei calcoli di attenzione su più GPU, riducendo il sovraccarico legato alla comunicazione tra i dispositivi. Questo approccio consente di massimizzare l'efficienza delle risorse hardware disponibili, garantendo che ogni GPU partecipi attivamente al processo senza inutili attese o rallentamenti. Inoltre, la riduzione degli accessi alla memoria HBM favorisce una maggiore

efficienza nei contesti di calcolo distribuito, rendendo FlashAttention una soluzione ideale per infrastrutture multi-nodo e per sistemi con risorse hardware limitate.

Un altro aspetto fondamentale è la capacità di FlashAttention di supportare lunghezze di sequenza maggiori rispetto agli approcci tradizionali. Grazie alla sua scalabilità, l'algoritmo permette di addestrare modelli che richiedono sequenze più lunghe, senza compromettere le prestazioni o esaurire la memoria disponibile. Questo vantaggio si traduce in un significativo miglioramento dell'accuratezza dei modelli, soprattutto in domini dove la gestione di informazioni a lungo termine è cruciale, come nei modelli di linguaggio naturale.

FlashAttention si distingue anche per la sua capacità di combinare calcolo e trasferimenti di memoria in modo integrato, riducendo al minimo il tempo complessivo di esecuzione. Questa caratteristica è particolarmente rilevante in scenari di calcolo parallelo, dove il bilanciamento del carico e la minimizzazione delle latenze di comunicazione rappresentano fattori critici per il successo.

Conclusioni

In questo lavoro abbiamo esplorato tecniche avanzate per l'addestramento distribuito, come ZeRO Redundancy Optimizer, Model Parallelism e Mixed Precision Training, dimostrando i vantaggi in termini di prestazioni e utilizzo della memoria. Inoltre, è stato introdotto l'Activation Checkpointing, una tecnica fondamentale per gestire l'addestramento di modelli di grandi dimensioni con risorse hardware limitate.

Prospettive future: Il prossimo lavoro si concentrerà sull'implementazione pratica delle tecniche descritte, utilizzando la libreria DeepSpeed su modelli avanzati come i Transformer. Questo includerà esperimenti concreti per ottimizzare i tempi di addestramento e massimizzare l'efficienza delle risorse.

Bibliografia

F. Gargiulo, A. Francesco Gentile, E. Greco. (2024, December). *Data Parallelism: Deep Learning con GPU Multiple, in modo efficiente e sostenibile*, RT-ICAR-NA-2024-04 https://intranet.icar.cnr.it/wp-content/uploads/2024/11/RT-ICAR-NA-2024-04.pdf

F. Gargiulo, A. Francesco Gentile, E. Greco. (2024, December). *Data Parallelism: Tecniche Avanzate di Stabilità e Convergenza di modelli su Larga Scala*, RT-ICAR-NA-2024-06
https://intranet.icar.cnr.it/wp-content/uploads/2024/11/RT-ICAR-NA-2024-06.pdf

NVIDIA. *NCCL Operations – Usage Guide*. NVIDIA Documentation, https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/operations.html. Accessed 20 June 2024.

PyTorch. *Getting Started with Distributed Data Parallel*. PyTorch Tutorials, https://pytorch.org/tutorials/intermediate/dist_tuto.html. Accessed 20 June 2024.

Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., Li, D., & He, Y. (2021). ZeRO-Offload: Democratizing Billion-Scale Model Training.

PyTorch. *Distributed Data Parallel*. PyTorch Documentation, https://pytorch.org/docs/stable/nn.html#torch.nn.parallel.DistributedDataParallel. Accessed 20 June 2024.

Microsoft and NVIDIA. *Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model*. 2021, https://developer.nvidia.com/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b/.

Li, Dong, et al. *PyTorch Distributed: Experiences on Accelerating*. Proceedings of the 2020 USENIX Conference on Operational Machine Learning (OpML '20), USENIX Association, 2020, https://www.usenix.org/conference/opml20/presentation/li.

PyTorch. *Distributed Communication Package* – torch.distributed. PyTorch Documentation, https://pytorch.org/docs/stable/distributed.html. Accessed 20 June 2024.

NVIDIA. Parallelism Strategies in NeMo Megatron. NVIDIA Documentation, https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/main/nlp/nemo_megatron/parallelisms.html. Accessed 20 June 2024.

NVIDIA. *Scaling Language Model Training to a Trillion Parameters Using Megatron*. NVIDIA Developer Blog, 21 Apr. 2021, https://developer.nvidia.com/blog/scaling-language-model-training-to-a-trillion-parameters-using-megatron/. Accessed 20 June 2024.

FairScale. *Pipeline Parallelism*. FairScale Documentation, https://fairscale.readthedocs.io/en/latest/deep_dive/pipeline_parallelism.html. Accessed 20 June 2024.

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V. A., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., Phanishayee, A., & Zaharia, M.

(2021). *Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM*. arXiv. https://arxiv.org/pdf/2104.04473.

Microsoft and NVIDIA. *Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model*. NVIDIA Developer Blog, 2021, https://developer.nvidia.com/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b/. Accessed 20 June 2024.

NVIDIA. *NVIDIA NeMo Framework Documentation*. NVIDIA, https://docs.nvidia.com/nemo-framework/. Accessed 20 June 2024.

Li, S., Xue, F., Baranwal, C., Li, Y., & You, Y. (2022). *Sequence Parallelism: Long Sequence Training from System Perspective*. arXiv preprint arXiv:2205.02620. https://arxiv.org/abs/2205.02620.

Korthikanti, V., Casper, J., Lym, S., McAfee, L., Andersch, M., Shoeybi, M., & Catanzaro, B. (2022). *Reducing Activation Recomputation in Large Transformer Models*. arXiv preprint arXiv:2205.05198. https://arxiv.org/abs/2205.05198.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., ... El Sayed, W. (2024). *Mixtral of Experts*. arXiv preprint arXiv:2401.04088. https://arxiv.org/pdf/2401.04088.pdf

Fedus, W., Zoph, B., & Shazeer, N. (2021). *Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity*. arXiv preprint arXiv:2101.03961. https://arxiv.org/abs/2101.03961.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., & Dean, J. (2017). *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In Proceedings of the 5th International Conference on Learning Representations* (ICLR 2017). https://arxiv.org/abs/1701.06538

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). *Adaptive mixtures of local experts*. Neural Computation, 3(1), 79–87. https://doi.org/10.1162/neco.1991.3.1.79

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., & Ré, C. (2022). *FlashAttention: Fast and memory-efficient exact attention with IO-awareness*. In Advances in Neural Information Processing Systems (NeurIPS 2022). https://arxiv.org/abs/2205.14135

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., & Wu, H. (2018). *Mixed precision training*. International Conference on Learning Representations (ICLR). Disponibile al link: https://arxiv.org/abs/1710.03740