Large Language Model Inference on Heterogeneous Clusters

Limits: 4 sec., 1024 MiB

Heterogeneous clusters are frequently utilized for deploying real-world large language model (LLM) inference tasks, a necessity arising from the varying acquisition times and diverse machine specifications. These clusters encompass computing units that vary in terms of computational power, memory size, memory bandwidth, and network communication capabilities. By carefully deploying the language model to the most appropriate machines and routing input queries to the optimal model instances, the full potential of the varying machine capabilities can be harnessed, thereby optimizing the overall efficiency and utilization of the cluster.

The task is to deploy large language model (LLM) inference in a heterogeneous cluster to achieve the lowest latency.

Suppose there is a heterogeneous cluster with n machines, numbered from 1 to n. The machine number i is described by five parameters:

- u_i — the number of computational units,
- f_i — the computational resources,
- d_i — the memory capacity,
- c_i — the memory read/write bandwidth,
- e_i — the communication bandwidth.

A large model is deployed in this cluster. The model is described with three parameters:

- l — the number of layers,
- h — the number of hidden dimensions,
- Φ — the number of parameters.

Each of the machines is internally split into pipelines. Each pipeline contains a single instance of the model. Let's denote the number of the pipelines for the machine number i as p_i.

Each pipeline has a unique index. The pipelines on the machine number 1 are numbered from 1 to p_1. The pipelines on the machine number 2 are numbered from $p_1 + 1$ to $p_1 + p_2$. The pipelines on the machine number i are numbered from $\sum_{j=1}^{i-1} p_j + 1$ to $\sum_{j=1}^{i} p_j$. The total number of pipelines on all machines is equal to $P = \sum_{i=1}^{n} p_i$.

Let's denote the index of the machine that the pipelines s is located on as $inst_s$.

The tensor parallel degree is a number of computational units dedicated to a single pipeline. For the machine number i this value should be uniform across all the pipelines. Let's denote this value as t_i. The following condition must hold: $p_i \cdot t_i = u_i$.

Let's denote the maximal batch size for the pipelines on the machine number i as b_i.

Suppose the cluster receives m query bursts. The j-th burst contains N_j queries. The intervals between bursts number j and $j+1$ is τ_j. For consistency, let's say that $\tau_m = 0$. The input and output lengths of the query number r in the burst number j are I_j^r and O_j^r, respectively.

Let's denote the index of the pipeline which will handle the request number r in the burst number j as g_j^r.

Let's define the following function:

$$H(x, y) = \begin{cases} 1, & \text{if } x = y \\ 0, & \text{otherwise} \end{cases}$$

The number of batches for processing the burst number j on the pipelines number s will be:

$$A_j^s = \left\lceil \frac{\sum_{r=1}^{N_j} H(g_j^r, s)}{b_{inst_s}} \right\rceil$$

Let's denote the index of the batch that the request number r in the burst number j will be processed in as W_j^r.

The task requirements are:

- decide the number of pipelines p_i, the tensor parallel degrees t_i and maximal batch sizes b_i for each machine,
- decide the allocation of the queries, i.e. the values of g_j^r and W_j^r for each request in each burst.

Your goal is to minimize three types of query inference latency: $L_{total}, L_{total}^{prefill}, L_{total}^{decode}$. Please refer to the Scoring section of this statement for more details.

Input

The first line of the input contains three integers l, h, Φ, which describe the model used in this test case. The second line contains three floating point number α, β, γ — the coefficients for the scoring function. The third line contains two integer numbers n and m — the number of machines and burst requests, respectively.

The next n lines describe the computing machines. Each of those line describes one computing machine and contain five integers $u_i, f_i', d_i', c_i', e_i'$, where u_i is a number of computational units of the machine, f_i', d_i', c_i', e_i' are the corresponging characteristics of the machine given in billions. To get the actual values, please just multiply the given numbers by 10^9: $f_i = f_i' \cdot 10^9$, $d_i = d_i' \cdot 10^9$, $c_i = c_i' \cdot 10^9$, $e_i = e_i' \cdot 10^9$.

Then the description of the burst requests follow. There are a total of m blocks, each block describes a single burst request. The first line of the description for the j-th burst request contains an integer N_j, followed by a floating point number τ_j. The next line contains N_j integers I_j^r. The next line contains N_j integers O_j^r.

Output

The first n lines should contain the number of pipelines, the tensor parallel degree and max batch sizes for the computing machines. The i-th of those lines should contain values p_i, t_i, b_i — the number of pipelines, tensor parallel degree and the max batch size for the machine number i. The number of pipelines and tensor parallel degree should satisfy the following condition: $p_i \cdot t_i = u_i$. The batch size should be up to 1000: $1 \leq b_i \leq 1000$.

The next m blocks should describe the burst requests. For burst request number j you should print N_j lines. The r-th of those lines should contain two integer values: g_i^r, W_j^r — the index of the pipeline which should handle the r-th request in the current burst request, and the batch index for this request. The pipelines are numbered from 1 to $\sum_{i=1}^n p_i$. The batches for pipeline number s and burst query number j are numbered from 1 to A_j^s.

Please note that the enumeration of batches is separate for each burst query. Also, for each pipeline s and each burst query j, each batch from 1 to $A_j^s - 1$ has to be complete, i.e. have exactly b_{inst_s} queries. The last batch number A_j^s may have b_{inst_s} or less queries.

Constraints

$30 \leq l \leq 100$,
$1000 \leq h \leq 15000$,
$3 \cdot 10^8 \leq \Phi \leq 2 \cdot 10^{11}$,
$0 \leq \alpha, \beta, \gamma \leq 1$,
$\alpha + \beta + \gamma = 1$,
$2 \leq n \leq 10$,
$2 \leq m \leq 100$,
$u_i = 8$,

$3 \cdot 10^{14} \le f_i \le 2.5 \cdot 10^{15}$,
$3.2 \cdot 10^{10} \le d_i \le 2 \cdot 10^{11}$,
$2 \cdot 10^{11} \le c_i \le 2 \cdot 10^{12}$,
$5 \cdot 10^{10} \le e_i \le 9 \cdot 10^{11}$,
$10 \le N_j \le 1000$,
$0.01 \le \tau_j \le 1 \; if \; j \ne m; \tau_j = 0 \; if \; j = m$,
$10 \le I_j^r, O_j^r \le 1000$,
each floating point number is given with exactly three digits after the decimal point.

Samples

Input (*stdin*)	Output (*stdout*)

```
32 4096 6700000000
0.856 0.066 0.078
5 2
8 156500 64 800 28
8 140000 32 400 28
8 131000 32 600 15
8 333500 80 1000 150
8 1000000 80 1500 225
10 0.180
250 290 275 322 166 299 334 301 269 369
224 152 274 221 199 291 128 194 195 148
10 0.000
318 248 243 305 267 303 266 234 393 257
237 203 210 245 241 137 199 200 163 184
```
	```
	1 8 1
	1 8 1
	1 8 1
	1 8 1
	1 8 1
	1 1
	2 1
	3 1
	4 1
	5 1
	1 2
	2 2
	3 2
	4 2
	5 2
	1 1
	2 1
	3 1
	4 1
	5 1
	1 2
	2 2
	3 2
	4 2
	5 2
	```

Notes

Scoring

Your output for a test is considered to be incorrect if any of the values in the output are invalid. Additionally, for each burst request j, each machine i should satisfy the memory capacity constraints:

$$d_i \ge d_i^{model} + d_{i,j}^{VCache}$$

$$d_i^{model} = \frac{2 \cdot \Phi}{t_i}$$

$$d_{i,j}^{VCache} = \frac{4 \cdot b_i \cdot l \cdot h \cdot \max\limits_{1 \le r \le N_j} (I_j^r + O_j^r)}{t_i}$$

If your output is incorrect, then the score for this testcase is 0.
Otherwise, your preliminary score is calculated as following:

$$score' = \left\lfloor \left\lfloor \frac{L_{opt} \cdot 10^7}{L_{total}} \right\rfloor \cdot \alpha + \left\lfloor \frac{L_{opt}^{prefill} \cdot 10^7}{L_{total}^{prefill}} \right\rfloor \cdot \beta + \left\lfloor \frac{L_{opt}^{decode} \cdot 10^7}{L_{total}^{decode}} \right\rfloor \cdot \gamma \right\rfloor$$

The values $L_{total}, L_{total}^{prefill}, L_{total}^{decode}$ represent the total latency, the prefilling stage total latency, and the decoding stage total latency between all pipelines and are calculated as following:

$$\begin{aligned} L_{total} &= \max_{1 \le s \le p} L_s \\ L_{total}^{prefill} &= \max_{1 \le s \le p} L_s^{prefill} \\ L_{total}^{decode} &= \max_{1 \le s \le p} L_s^{decode} \end{aligned}$$

Where the values $L_s, L_s^{prefill}, L_s^{decode}$ are respective values for a specific pipeline:

$$\begin{aligned} L_s &= \sum_{j=1}^{m} L_{s,j} \\ L_s^{prefill} &= \sum_{j=1}^{m} L_{s,j}^{prefill} \\ L_s^{decode} &= \sum_{j=1}^{m} (L_{s,j}^{decode\_comp} + L_{s,j}^{decode\_mem\_comm}) \end{aligned}$$

The values $L_{s,j}, L_{s,j}^{prefill}, L_{s,j}^{decode\_comp}, L_{s,j}^{decode\_mem\_comm}$ represent the total latency, the prefilling latency, the decoding latency, and the decoding stage memory communication latency for the pipeline s to process the query burst j, respectively.

$$L_{s,j} = \max\left(L_{s,j}^{prefill} + L_{s,j}^{decode\_comp} + L_{s,j}^{decode\_mem\_comm} + L_{s,j}^{comm}, \tau_j\right)$$

Where $L_{s,j}^{comm}$ represent the communication latency for the pipeline s to process the query burst j, respectively
The latencies are calculated as follows:

$$\begin{aligned} L_{s,j}^{prefill} &= \frac{2 \cdot \Phi}{t_{inst_s}} \cdot \frac{\sum_{r=1}^{N_j} I_j^r \cdot H(g_j^r, s)}{f_{inst_s}} \\ L_{s,j}^{decode\_comp} &= \frac{2 \cdot \Phi}{t_{inst_s}} \cdot \frac{\sum_{r=1}^{N_j} O_j^r \cdot H(g_j^r, s)}{f_{inst_s}} \\ L_{s,j}^{decode\_mem\_comm} &= \frac{2 \cdot \Phi + 8 \cdot l \cdot h \cdot (v_{s,j}^{accu} + v_{s,j}^{last\_batch})}{t_{inst_s} \cdot c_{inst_s}} \end{aligned}$$

Please note that the following two equations have been changes since the start of the competition. You can download the updated scorer source code using the link below.

$$v_{s,j}^{accu} = \begin{cases} b_{inst_s} \cdot \sum_{x=1}^{A_j^s - 1} \max\limits_{1 \le r \le N_j} \left(O_j^r \cdot (I_j^r + \frac{1}{2}(O_j^r - 1)) \cdot H(W_j^r, x) \cdot H(g_j^r, s) \right) & \text{, if } A_j^s > 1 \\ 0 & \text{, otherwise} \end{cases}$$

$$v_{s,j}^{last\_batch} = \sum_{r=1}^{N_j} \left(H(W_j^r, A_j^s) \cdot H(g_j^r, s) \right) \cdot \max_{1 \le r \le N_j} \left(O_j^r \cdot (I_j^r + \frac{1}{2}(O_j^r - 1)) \cdot H(W_j^r, A_j^s) \cdot H(g_j^r, s) \right)$$

$$L_{s,j}^{comm} = \frac{8 \cdot l \cdot h \cdot (v_{s,j}^{accu} + v_{s,j}^{last\_batch})}{e_{inst_s}} \cdot \frac{t_{inst_s} - 1}{t_{inst_s}}$$

The values $L_{opt}, L_{opt}^{prefill}, L_{opt}^{decode}$ represent the minimum latencies under an unattainable optimal scenario.

$$\begin{aligned}
L_{opt} &= \frac{2 \cdot \Phi}{u^{max}} \cdot \frac{m \cdot N^{min} \cdot IO^{min}}{u^{max} \cdot f^{max}} \\
L_{opt}^{prefill} &= \frac{2 \cdot \Phi}{u^{max}} \cdot \frac{m \cdot N^{min} \cdot I^{min}}{u^{max} \cdot f^{max}} \\
L_{opt}^{decode} &= \frac{2 \cdot \Phi}{u^{max}} \cdot \frac{m \cdot N^{min} \cdot O^{min}}{u^{max} \cdot f^{max}}
\end{aligned}$$

Where:

$$\begin{aligned}
u^{max} &= \max_{1 \le i \le n} u_i \\
f^{max} &= \max_{1 \le i \le n} f_i \\
N^{min} &= \min_{1 \le j \le m} N_j \\
IO^{min} &= \min_{1 \le j \le m, 1 \le r \le N_j} (I_j^r + O_j^r) \\
I^{min} &= \min_{1 \le j \le m, 1 \le r \le N_j} I_j^r \\
O^{min} &= \min_{1 \le j \le m, 1 \le r \le N_j} O_j^r
\end{aligned}$$

Penalties

The generation of the first token and incremental tokens must satisfy the proposed time constraints, otherwise, your score will get a penalty:

$$\begin{aligned}
L_{first\_token} &= \frac{L_{total}^{prefill} \cdot P}{\sum_{j=1}^{m} N_j} \\
L_{incremental\_token} &= \frac{L_{total}^{prefill} \cdot P}{\sum_{j=1}^{m} \sum_{r=1}^{N_j} O_j^r}
\end{aligned}$$

$$L_{first\_token\_threshold} = 1.000$$
$$L_{incremental\_token\_threshold} = 0.050$$

The first token penalty is equal to:

$$pen_{first} = \min\left(\frac{L_{first\_token\_threshold}}{L_{first\_token}}, 1\right)$$

The incremental penalty is equal to:

$$pen_{incremental} = \min\left(\frac{L_{incremental\_token\_threshold}}{L_{incremental\_token}}, 1\right)$$

The final score is calculated as follows:

$$score = \lfloor score' \cdot pen_{first} \cdot pen_{incremental} \rfloor$$

Submissions

- The execution time limit is 4 seconds per test case, and the memory limit is 1024 mebibytes.
- The code size limit is 64 kibibytes.
- The compilation time limit is 1 minute.
- There are 50 provisional test cases. Your submissions will be evaluated on the provisional set during the submission phase.
- You can submit your code once every 10 minutes, and you will get feedback with your score for each of the provisional tests.
- There will be 500 test cases in the final testing after the submission phase is over. Please note that provisional tests are **not included** in the final testing. The final results will be announced in one week.

Quick start

Check the sample solution, which reads the data, handles the requests in a roud-robin fashion, and prints the results.

- C++
- Python
- C#

Tests

All test cases, including provisional and final sets, are generated by the generator. Please note that the generator uses an Algotester Generator Library. You can use the generator for local testing by taking a few simple steps:

1. Save the library source code to a file named `algotester_generator.h`
2. Save the generator source code to a file named `generator.cpp`
3. Compile the source code:

   ```
   g++ generator.cpp -O2 -o generator
   ```

4. Generate a test case

   ```
   ./generator <seed>
   ```

   This will print a test case for a specific seed to *standard output*.

Checker and scorer

The checker verifies the correctness of your output for the test. It checks if the data in your output file if valid and if you pass the memory capacity constraints described in the Scoring section. If there is an issue with your solution the checker prints a message which describes the problem to *standard output*. If your output is correct, the checker prints nothing.

The scorer calculates the score of your output. Please note that the scorer doesn't check for the validity of the output and assumes that the output is correct, i.e. the checker was run prior to the scorer and didn't report any issues.

The checker and the scorer both use an Algotester Library. You can use the checker and the scorer for local testing by taking a few simple steps:

1. Save the library source code to a file named `algotester.h`
2. Save the checker source code to a file named `checker.cpp`
3. Save the scorer source code to a file named `scorer.cpp`
4. Compile the source codes:

   ```
   g++ checker.cpp -O2 -o checker
   g++ scorer.cpp -O2 -o scorer
   ```

5. Run the checker and provide it with the test case and your output for this test case:

   ```
   ./checker <input-file> <your-output-file>
   ```

   This will report any issues with your output for the test case to *standard output.*
6. Run the scorer and provide it with the test case and your output for this test case:

   ```
   ./scorer <input-file> <your-output-file>
   ```

   This will print your score for the test case to *standard output.*