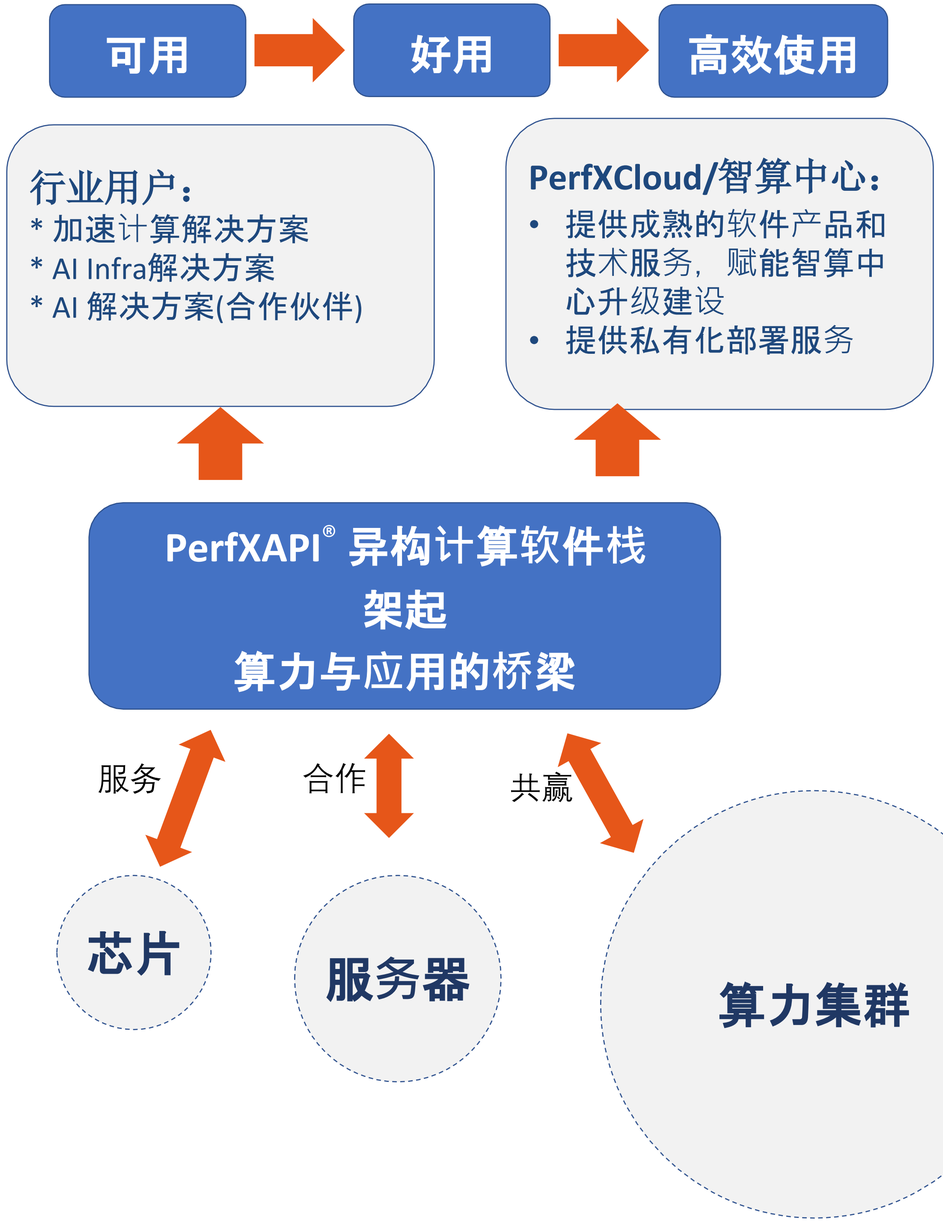
PerfXLM 2.0:大模型推理引擎进展

张先轶
澎峰科技
xianyi@perfxlab.com

关于澎峰科技

- 2016年，澎峰科技（PerfXLab)成立，核心团队来自中科院
- 公司一直致力于研发**算力基础软件及AI解决方案**（高性能计算库、异构计算框架以及软硬融合解决方案等），为算力芯片和算力应用行业加速计算解决方案（华为，燧原，平头哥，华大九天，中船等）



公司主要获奖：

- 2016年，中国计算机学会科技进步二等奖
- 2017年，中国科学院杰出科技成就奖
- 2018年，北京雏鹰人才计划，国家高新企业
- 2021年，数字中国·集成电路赛道特等奖
- 2021年，创芯中国·决赛一等奖
- 2021年，CRVA联盟，软件工作组副组长单位
- 2022年，OpenCAX SIG10组长单位
- 2022年11月15日，ChinaSC中国超级算力大会荣获 “算力软件基建领军企业”和“中国智能计算卓越贡献奖”双项荣誉
- 2023年，入选北京市“专精特新”中小企业
- 2023年，入选中国互联网协会算网云协同系统工作委员会成员单位
- 2023年，OpenBLAS获得全球开源贡献Open100
- 2023年，北京市自然科学二等奖

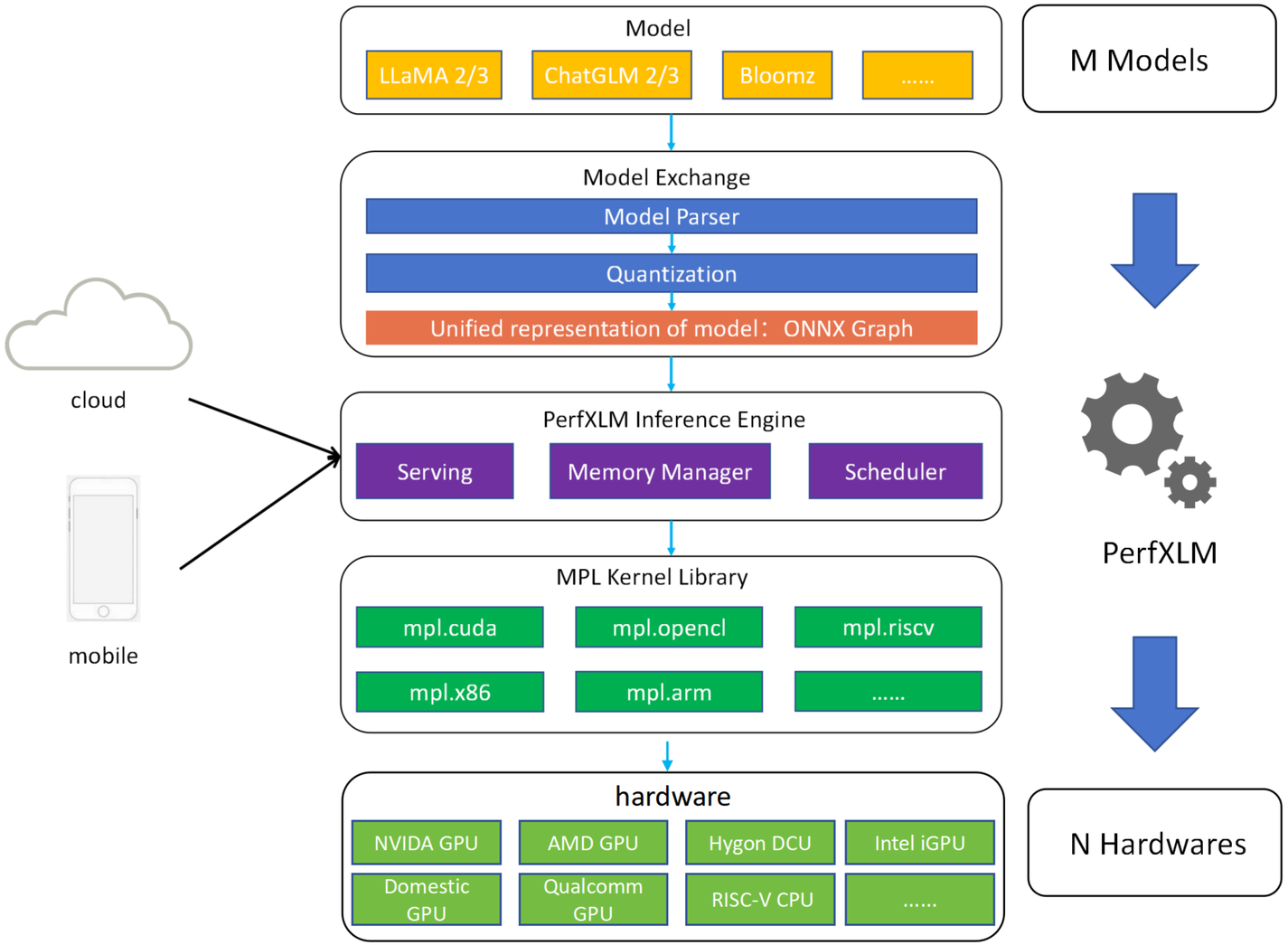
PerfXLM自研大模型推理引擎

PerfXLab

云端和边缘端一体
- 支持云端GPU/NPU
- 支持嵌入式GPU/NPU

支持多种硬件
- 国际主流硬件：NVIDIA GPU，Intel iGPU，高通SoC等
- 国产GPU/NPU硬件：RISC-V CPU, Hygon DCU，燧原等。

深入性能优化
- 多种算子融合与优化技术
- 核心计算库优化等
- 量化技术
- 多卡并行
- 内存优化



2. PerfxLM介绍

PerfxLM 使用接口

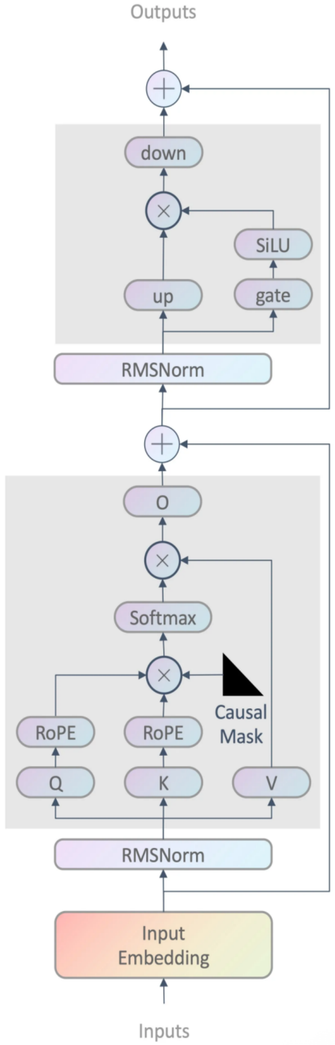
```python
from vllm import LLM, SamplingParams

# Sample prompts.
prompts = [
    "Hello, my name is",
    "The president of the United States is",
    "The capital of France is",
    "The future of AI is",
]
# Create a sampling params object.
sampling_params = SamplingParams(temperature=0.8, top_p=0.95)

# Create an LLM.
llm = LLM(model="facebook/opt-125m")
# Generate texts from the prompts. The output is a list of RequestOutput objects
# that contain the prompt, generated text, and other information.
outputs = llm.generate(prompts, sampling_params)
# Print the outputs.
for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
```

```python
from perfxlm import LLM, SamplingParams

# Sample prompts.
prompts = [
    "Hello, my name is",
    "The president of the United States is",
    "The capital of France is",
    "The future of AI is",
]
# Create a sampling params object.
sampling_params = SamplingParams(temperature=0.8, top_p=0.95)

# Create an LLM.
llm = LLM(model="facebook/opt-125m")
# Generate texts from the prompts. The output is a list of RequestOutput objects
# that contain the prompt, generated text, and other information.
outputs = llm.generate(prompts, sampling_params)
# Print the outputs.
for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
```

用户只需将import vllm改成import perfxlm就能获得与vllm一致的使用体验，
减少用户迁移框架的成本。

2. PerfxLM介绍

PerfxLM 调用逻辑: 第1步通过Python API组网



LLaMA 模型

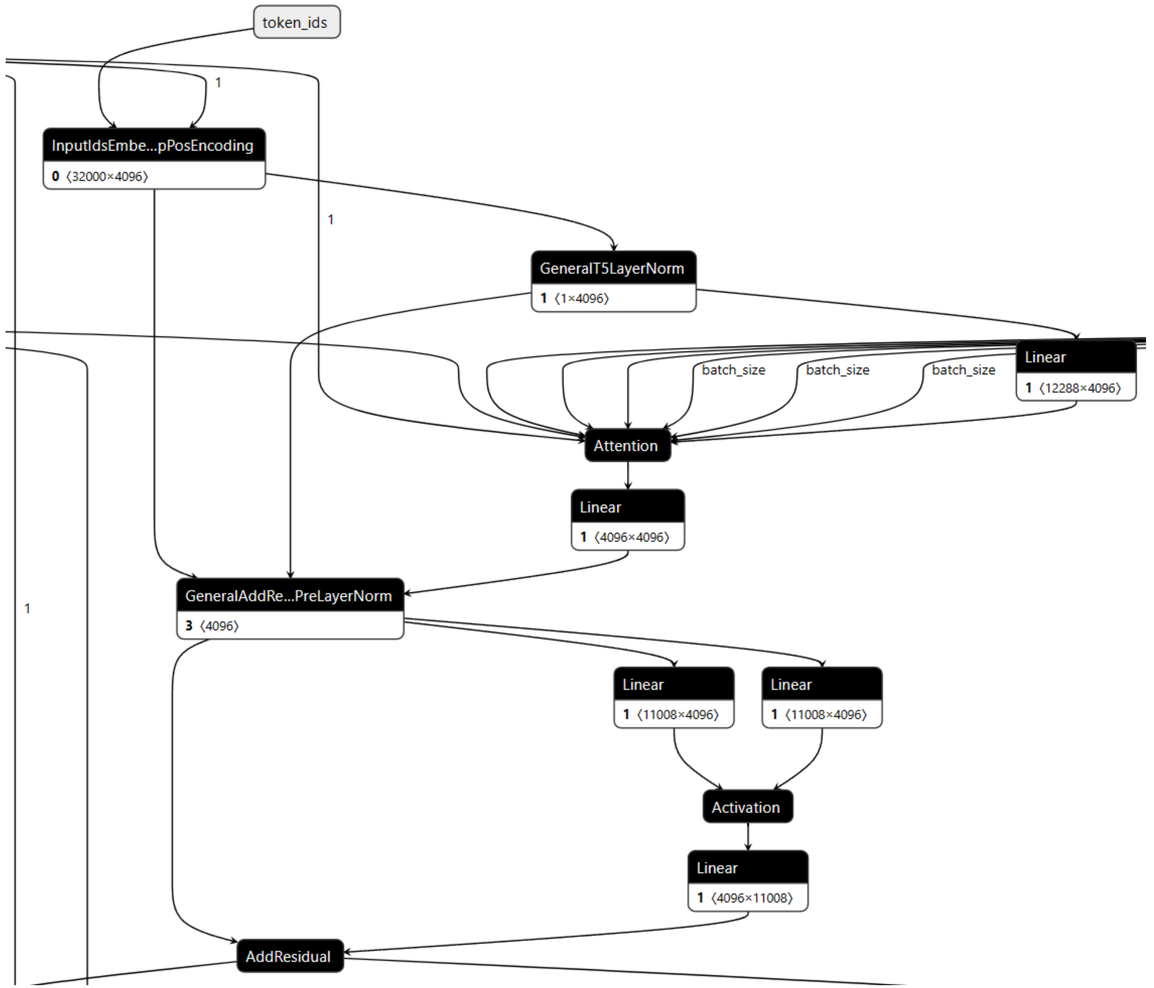
```python
class CacheAttention(torch.nn.Module):
    def __init__(self, args: ModelParams,layer_idx: int = 0):...
    def forward(self, x: torch.Tensor, mask: torch.Tensor,...

class FeedForward(nn.Module):
    # without MoE Mode, ia3.
    # int8_mode = 0.
    # without MoE Mode, ia3, gated_activation.
    # int8_mode = 0
    def __init__(...
    ):...

    def forward(self, x: torch.Tensor):...

class TransformerBlock(nn.Module):
    def __init__(self, args: ModelParams, layer_id: int):...

    def forward(self, x: torch.Tensor,mask:torch.Tensor, masked_t

class Transformer(nn.Module):
    def __init__(self, params: ModelParams):...


    @torch.inference_mode()
    def forward(self,tokens: torch.Tensor,step: torch.Tensor,...
```

纯Python代码、Torch风格构建模型

2. PerfxLM介绍

PerfxLM 调用逻辑: 第2步解析python代码，导出onnx图

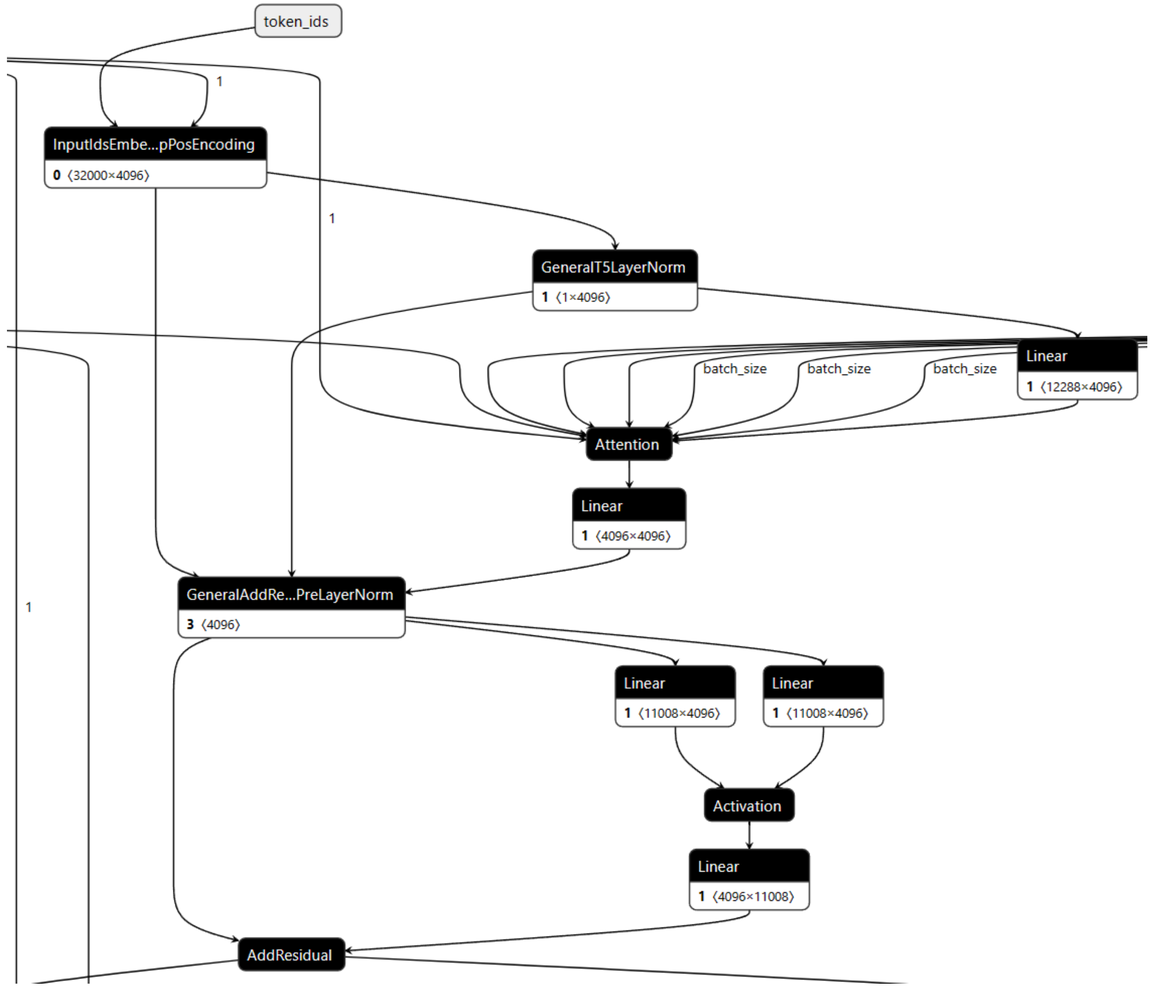
```python
class CacheAttention(torch.nn.Module):
    def __init__(self, args: ModelParams,layer_idx: int = 0):...
    def forward(self, x: torch.Tensor, mask: torch.Tensor,...

class FeedForward(nn.Module):
    # without MoE Mode, ia3.
    # int8_mode = 0.
    # without MoE Mode, ia3, gated_activation.
    # int8_mode = 0
    def __init__(...
    ):...

    def forward(self, x: torch.Tensor):...

class TransformerBlock(nn.Module):
    def __init__(self, args: ModelParams, layer_id: int):...

    def forward(self, x: torch.Tensor,mask:torch.Tensor, masked_t

class Transformer(nn.Module):
    def __init__(self, params: ModelParams):...


    @torch.inference_mode()
    def forward(self,tokens: torch.Tensor,step: torch.Tensor,...
```



对于llama模型为例，只有7个OP，易于维护并且可以针对硬件定制优化。

2. PerfxLM介绍

PerfxLM 调用逻辑: 第3步解析ONNX图，并在引擎层进行调度



PerfxLM推理引擎层

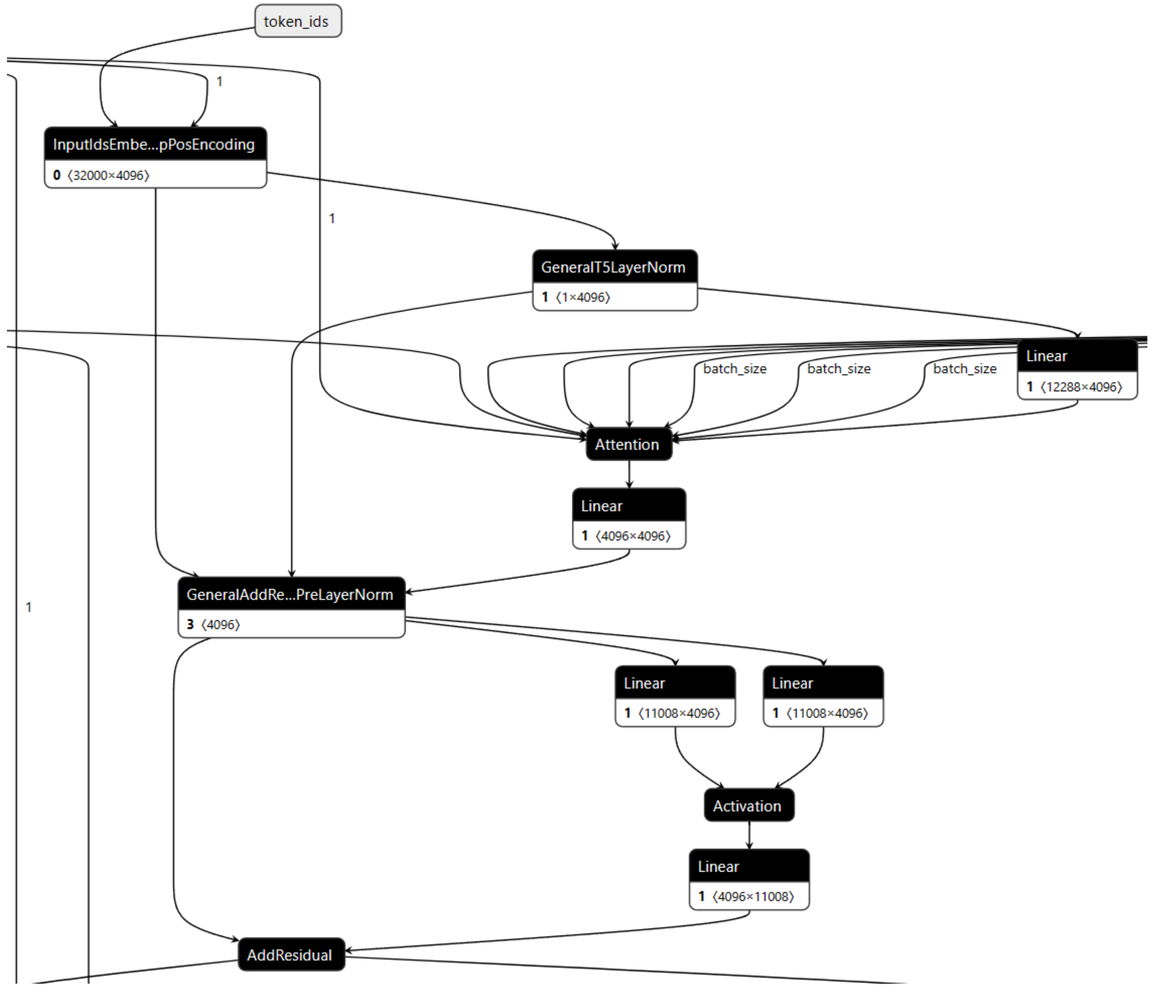
- Continuous Batching
- Prefix Caching
- Page Attention

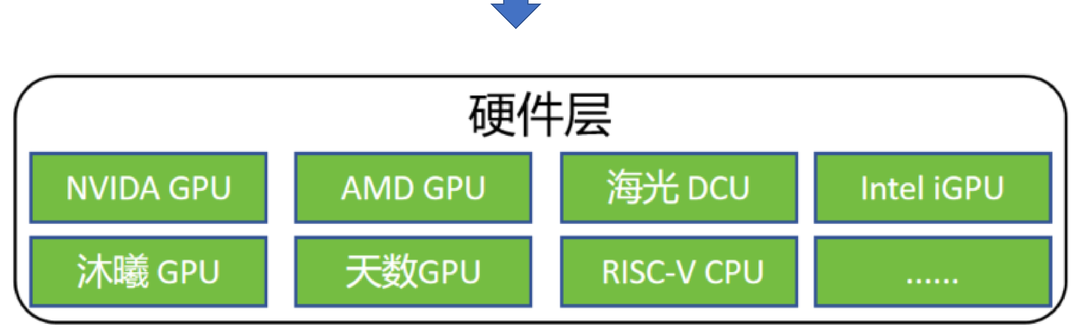
针对硬件的核心算子优化

在推理引擎层提供了完善的调度算法实现

PerfxLM介绍

PerfxLM 调用逻辑: 第4步针对硬件的定制算子优化



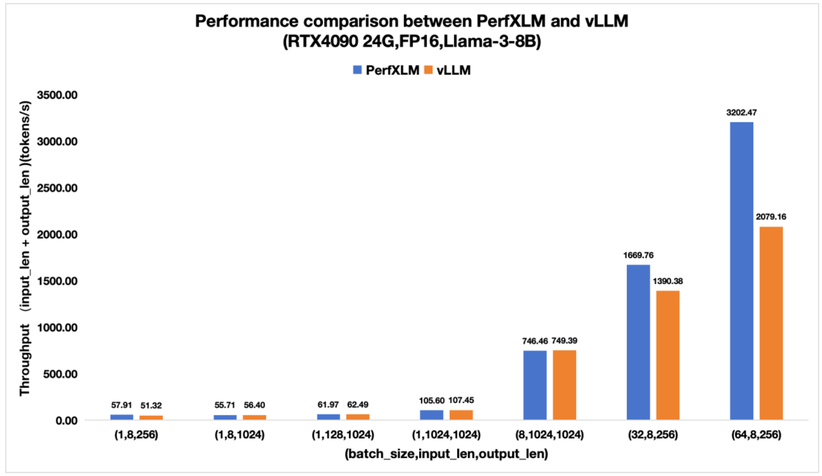


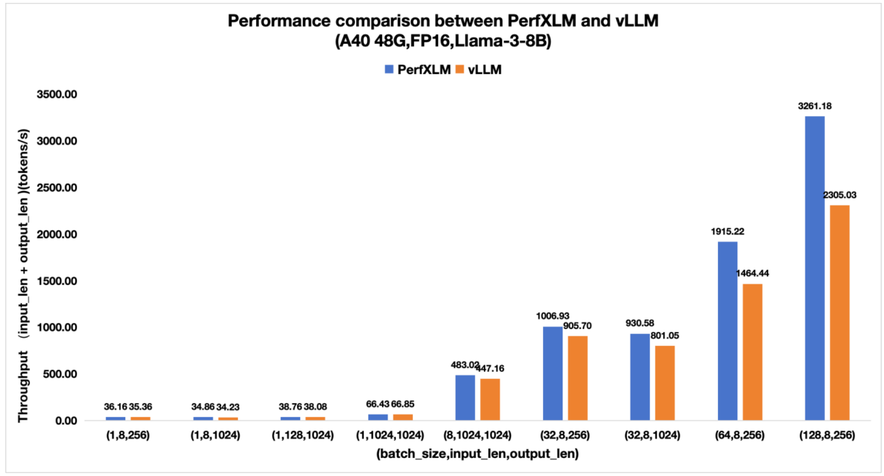
支持了各种编程语言以及大量的硬件

PerfXLM性能结果

NVIDIA平台初步结果

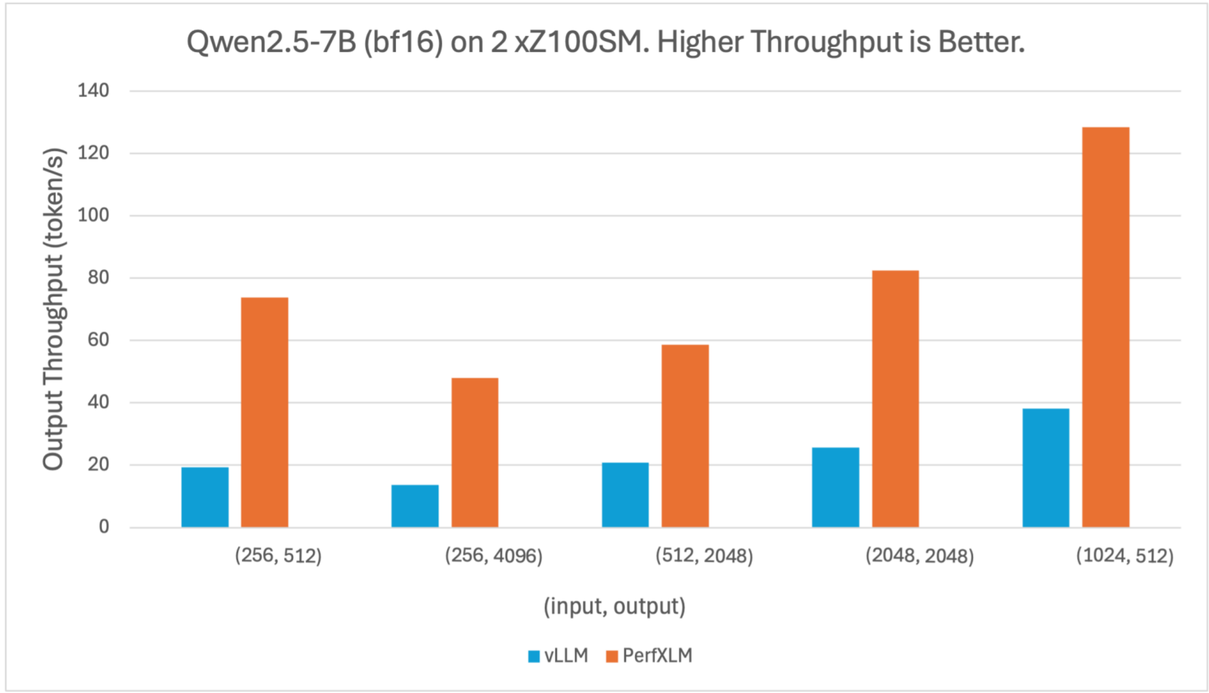
- Llama3 8B
- RTX4090和A40





DCU平台结果

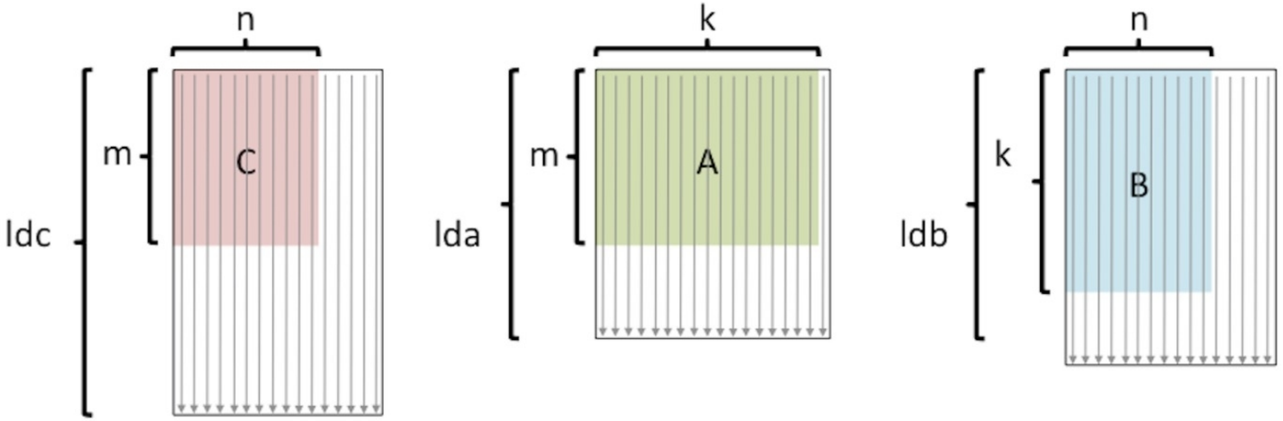
- Qwen2.5 7B fp16



PerfXLM on RISC-V CPU

GEMM optimization for prefill stage:

Optimize OpenBLAS by RISC-V Vector 1.0

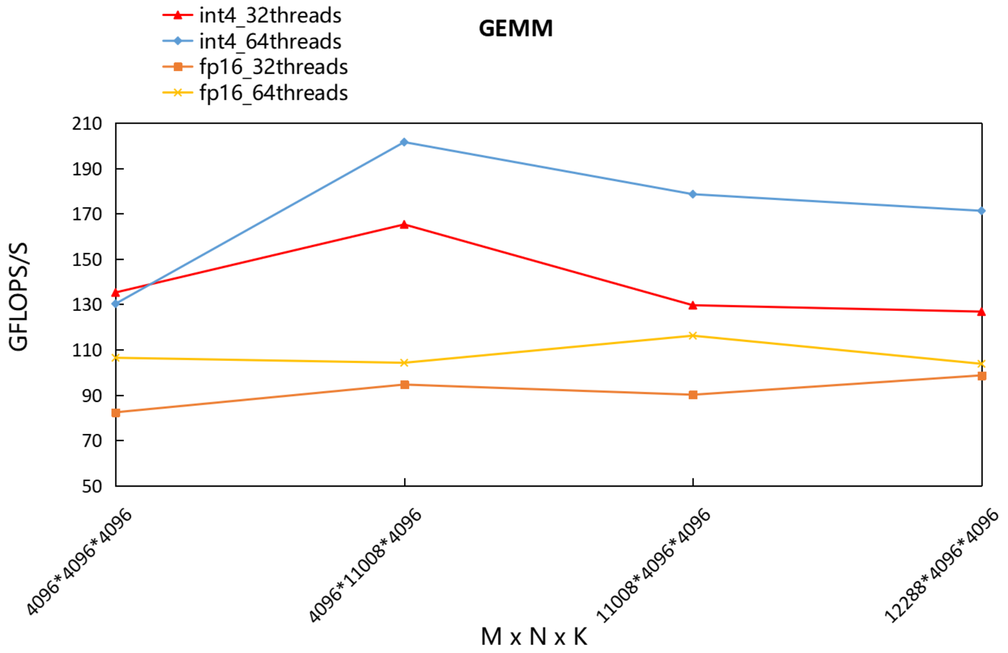



```
#define KERNEL16x4_I \
        "addi       t1,    %[PB], 1*4  \n\t"\
        "addi       t2,    %[PB], 2*4  \n\t"\
        "addi       t3,    %[PB], 3*4  \n\t"\
        "flw        ft0,  (%[PB])      \n\t"\
        "flw        ft1,  (t1)         \n\t"\
        "flw        ft2,  (t2)         \n\t"\
        "flw        ft3,  (t3)         \n\t"\
        "vle.v      v0,   (%[PA])      \n\t"\
        "addi       t4,    %[PA], 4*4  \n\t"\
        "addi       t5,    %[PA], 8*4  \n\t"\
        "vfmv.v.f   v8,   ft0          \n\t"\
        "addi       t6,    %[PA], 12*4  \n\t"\
        "addi       %[PA], %[PA], 16*4  \n\t"\
        "vle.v      v1,   (t4)         \n\t"\
        "addi       t4,    t4,    16*4  \n\t"\
        "vfmv.v.f   v9,   ft1          \n\t"\
        "vle.v      v2,   (t5)         \n\t"\
        "addi       t5,    t5,    16*4  \n\t"\
        "vle.v      v3,   (t6)         \n\t"\
        "addi       t6,    t6,    16*4  \n\t"\
        "vfmv.v.f   v10,  ft2          \n\t"\
        "addi       %[PB], %[PB], 4*4  \n\t"\
        "vle.v      v4,   (%[PA])      \n\t"\
        "addi       %[PA], %[PA], 16*4  \n\t"\
        "vfmv.v.f   v11,  ft3          \n\t"\
        "vfmacc.vv  v16,  v8,    v0   \n\t"\
        "addi       t1,   t1,     4*4  \n\t"\
        "vle.v      v5,   (t4)         \n\t"\
        "addi       t4,    t4,    16*4  \n\t"\
        "vfmacc.vv  v17,  v8,    v1   \n\t"\
```

PerfXLM on RISC-V CPU

GEMM optimization for prefill stage:

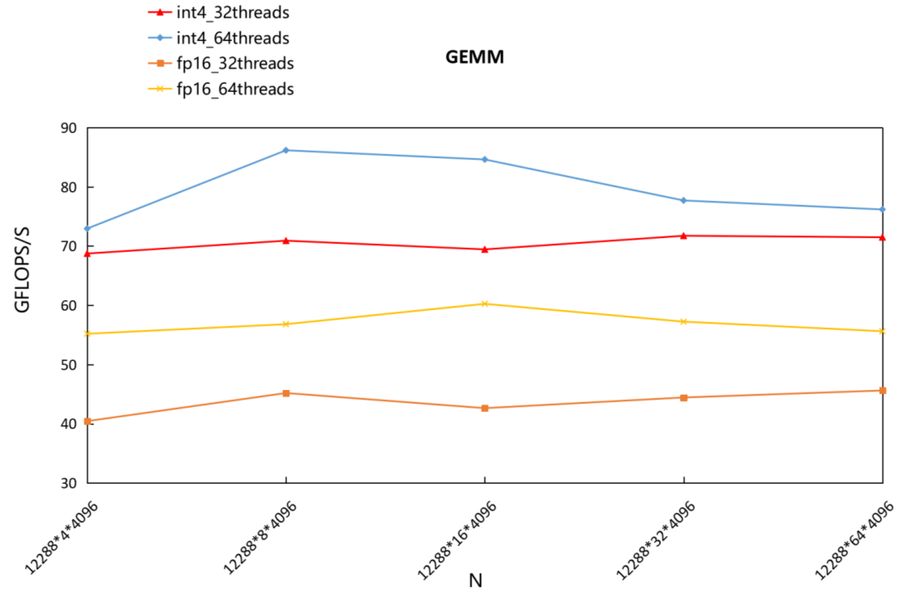
- Single batch: tall-and-skinny matrix
- Vector instruction optimization
- GEMM blocking
- Multi-threading



GEMM Performance (Big and square matrices)

RISC-V CPUs Testbed

- SG2044 CPU
  - Xuantie C920 2.0GHz
    - RVV 0.7.1, 128-bit
    - L1-D 64KB
  - 4 cores/cluster
    - Shared L2 1MB
  - 64 cores (16 cluster)
    - Shared L3 64MB
- **4x** DDR4 memory controllers

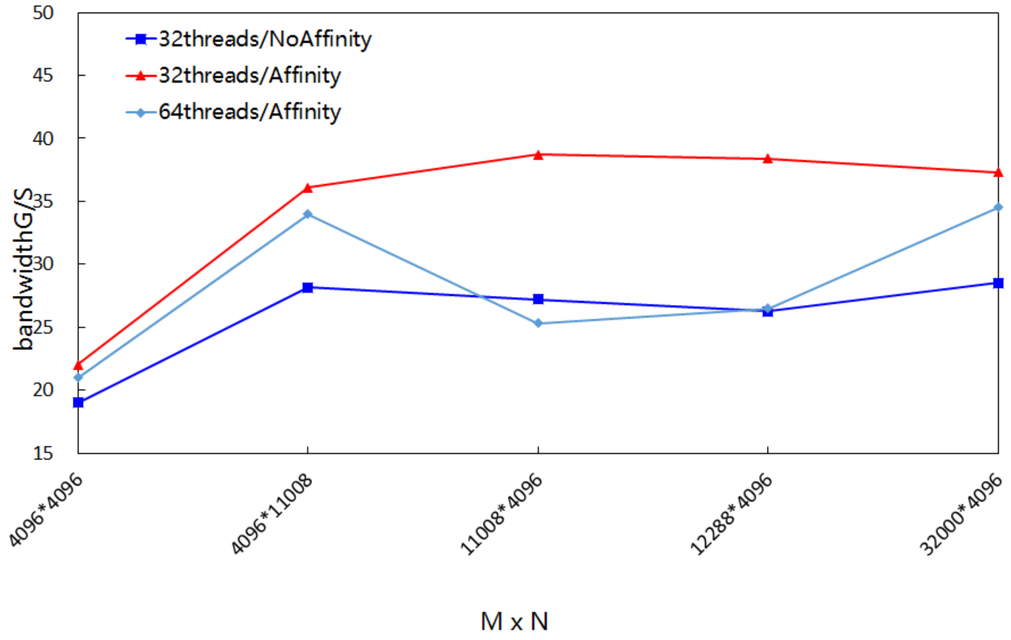


GEMM Performance (tall-and-skinny matrices)

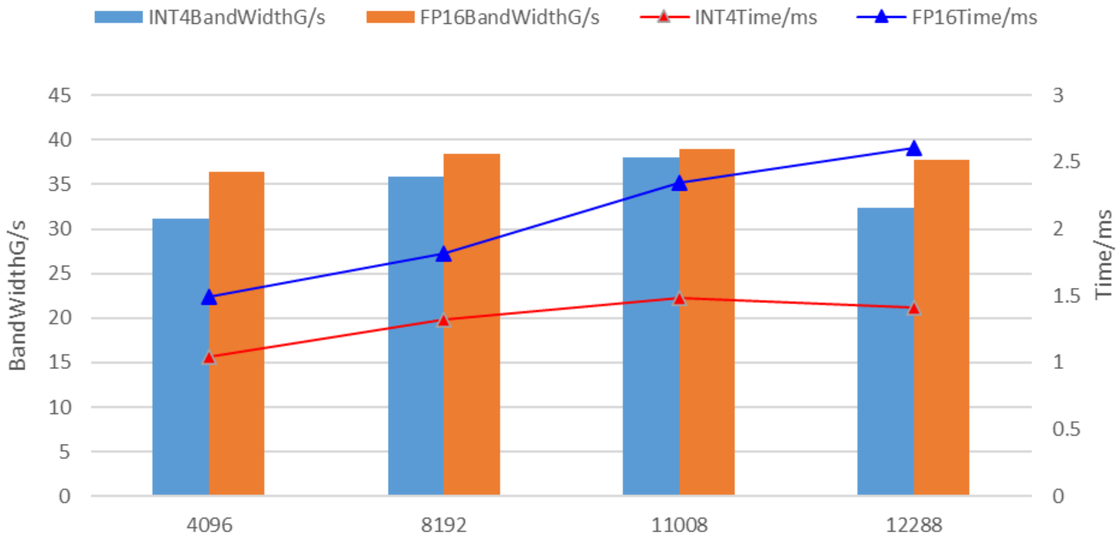
PerfXLM on RISC-V CPU

GEMV optimization for decoding stage:

- Vector instruction optimization



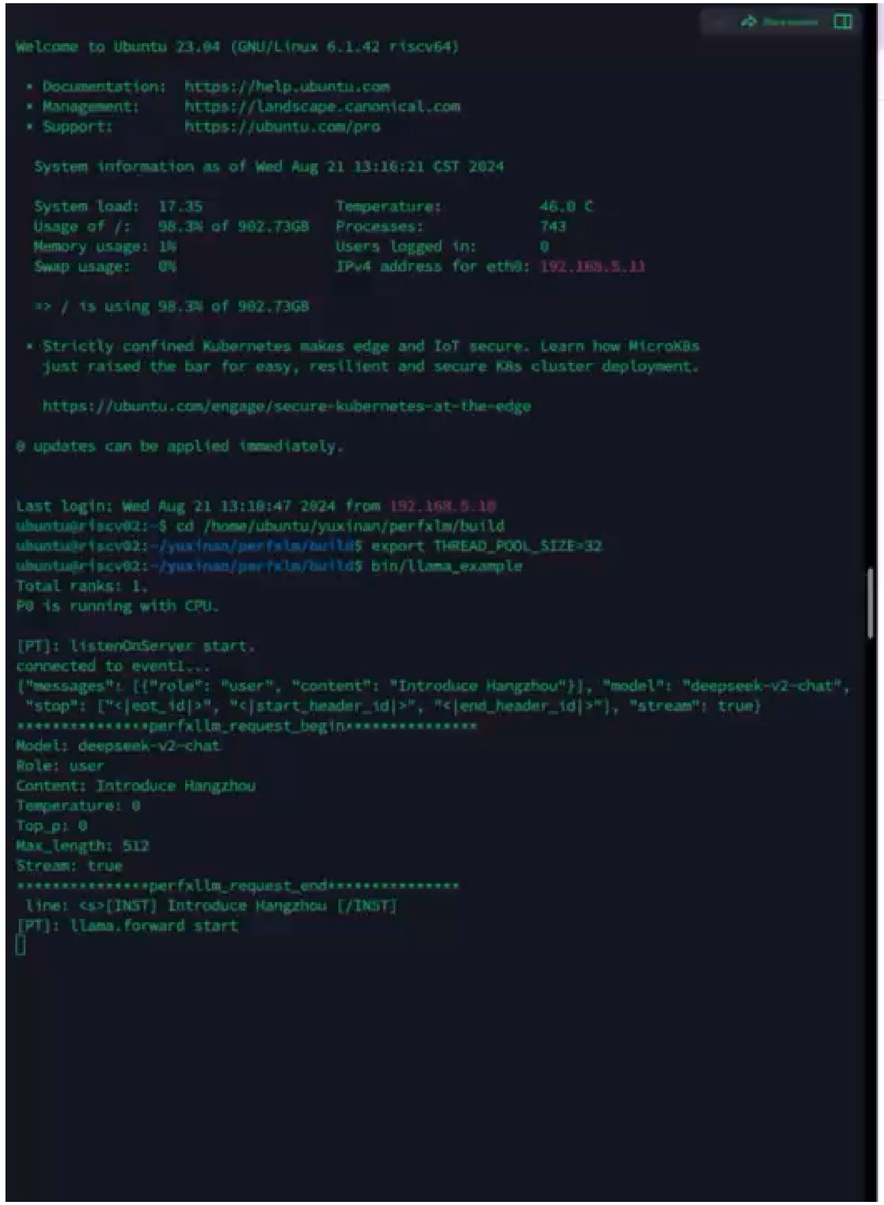
GEMV performance under different number of threads and affinity.



GEMV Performance (32 of threads)

Performance of the GEMV operator after optimization. The input of the GEMV performance test is: n =4096, m changes.

RISC-V平台的PerfXLM ChatBot应用案例



总结&展望

PerfXLab

PerfXLM 2.0

- SG2044
- RVV 1.0
- 多核

PerfXLM+PerfXCloud+RISC-V的机会

- 模型推理和微调平台，API兼容接口，MAAS模式
- 用户替换0成本