Driving vLLM Inference Performance on Intel® Xeon®

Presenters: Tony Wu, Jiang Li

Intel China

April 2024

InfoQ 极客传媒

已结束 北京站 2000 人

QCon 全球软件开发大会 暨智能软件开发生态展

架构演进 | AIGC
研发效能 | 向量数据库

6 月 14–15 日 深圳站 1000 人

ArchSummit 全球架构师峰会 暨智能软件开发生态展

业务架构 | 高可用
AI 架构 | 架构师成长

10 月 18–19 日 上海站 1500 人

QCon 全球软件开发大会 暨智能软件开发生态展

AIGC | AI 原生应用
软件工程 | 金融科技

4月 5月 6月 8月 10月 12月

5 月 17–18 日 北京站 2000 人

AiCon 全球人工智能开发与应用大会 暨大模型应用生态展

通用大模型 | AI Agent
LLMOps | AI Infra

8 月 16–17 日 上海站 1000 人

FCon 全球金融科技大会 暨智能软件开发生态展

金融大模型 | 核心系统
数字化转型 | 数据安全

12 月 20–21 日 北京站 1000 人

ArchSummit 全球架构师峰会 暨智能软件开发生态展

未来架构 | 高可用高并发
AI 应用 | 架构师思维

极客邦科技 2024 年会议规划

促进软件开发及相关领域知识与创新的传播

访问大会官网

参会咨询

LEGAL DISCLAIMERS

- INTEL TECHNOLOGIES' FEATURES AND BENEFITS DEPEND ON SYSTEM CONFIGURATION AND MAY REQUIRE ENABLED HARDWARE, SOFTWARE OR SERVICE ACTIVATION. PERFORMANCE VARIES DEPENDING ON SYSTEM CONFIGURATION. NO COMPUTER SYSTEM CAN BE ABSOLUTELY SECURE. CHECK WITH YOUR SYSTEM MANUFACTURER OR RETAILER OR LEARN MORE AT [INTEL.COM].
- FOR MORE COMPLETE INFORMATION ABOUT PERFORMANCE AND BENCHMARK RESULTS, VISIT WWW.INTEL.COM/BENCHMARKS.
- SOFTWARE AND WORKLOADS USED IN PERFORMANCE TESTS MAY HAVE BEEN OPTIMIZED FOR PERFORMANCE ONLY ON INTEL MICROPROCESSORS.

  PERFORMANCE TESTS, SUCH AS SYSMARK AND MOBILEMARK, ARE MEASURED USING SPECIFIC COMPUTER SYSTEMS, COMPONENTS, SOFTWARE, OPERATIONS AND FUNCTIONS. ANY CHANGE TO ANY OF THOSE FACTORS MAY CAUSE THE RESULTS TO VARY. YOU SHOULD CONSULT OTHER INFORMATION AND PERFORMANCE TESTS TO ASSIST YOU IN FULLY EVALUATING YOUR CONTEMPLATED PURCHASES, INCLUDING THE PERFORMANCE OF THAT PRODUCT WHEN COMBINED WITH OTHER PRODUCTS. FOR MORE COMPLETE INFORMATION VISIT WWW.INTEL.COM/BENCHMARKS.
- TESTS DOCUMENT PERFORMANCE OF COMPONENTS ON A PARTICULAR TEST, IN SPECIFIC SYSTEMS. DIFFERENCES IN HARDWARE, SOFTWARE, OR CONFIGURATION WILL AFFECT ACTUAL PERFORMANCE. CONSULT OTHER SOURCES OF INFORMATION TO EVALUATE PERFORMANCE AS YOU CONSIDER YOUR PURCHASE. FOR MORE COMPLETE INFORMATION ABOUT PERFORMANCE AND BENCHMARK RESULTS, VISIT WWW.INTEL.COM/BENCHMARKS
- COST REDUCTION SCENARIOS DESCRIBED ARE INTENDED AS EXAMPLES OF HOW A GIVEN INTEL- BASED PRODUCT, IN THE SPECIFIED CIRCUMSTANCES AND CONFIGURATIONS, MAY AFFECT FUTURE COSTS AND PROVIDE COST SAVINGS. CIRCUMSTANCES WILL VARY. INTEL DOES NOT GUARANTEE ANY COSTS OR COST REDUCTION.
- RESULTS HAVE BEEN ESTIMATED OR SIMULATED USING INTERNAL INTEL ANALYSIS OR ARCHITECTURE SIMULATION OR MODELING, AND PROVIDED TO YOU FOR INFORMATIONAL PURPOSES. ANY DIFFERENCES IN YOUR SYSTEM HARDWARE, SOFTWARE OR CONFIGURATION MAY AFFECT YOUR ACTUAL PERFORMANCE.
- INTEL DOES NOT CONTROL OR AUDIT THIRD-PARTY BENCHMARK DATA OR THE WEB SITES REFERENCED IN THIS DOCUMENT. YOU SHOULD VISIT THE REFERENCED WEB SITE AND CONFIRM WHETHER REFERENCED DATA ARE ACCURATE.
- INTEL PROCESSORS OF THE SAME SKU MAY VARY IN FREQUENCY OR POWER AS A RESULT OF NATURAL VARIABILITY IN THE PRODUCTION PROCESS.
- ALL INFORMATION PROVIDED HERE IS SUBJECT TO CHANGE WITHOUT NOTICE. CONTACT YOUR INTEL REPRESENTATIVE TO OBTAIN THE LATEST INTEL PRODUCT SPECIFICATIONS AND ROADMAPS.
- NO COMPUTER SYSTEM CAN BE ABSOLUTELY SECURE.
- INTEL, THE INTEL LOGO, XEON, INTEL VPRO, INTEL XEON PHI, LOOK INSIDE., ARE TRADEMARKS OF INTEL CORPORATION IN THE U.S. AND/OR OTHER COUNTRIES.
- *OTHER NAMES AND BRANDS MAY BE CLAIMED AS THE PROPERTY OF OTHERS.
- MICROSOFT, WINDOWS, AND THE WINDOWS LOGO ARE TRADEMARKS, OR REGISTERED TRADEMARKS OF MICROSOFT CORPORATION IN THE UNITED STATES AND/OR OTHER COUNTRIES.
- © 2024 INTEL CORPORATION.

Executive Summary

- vLLM 是業界领先的开源服务框架
- Intel® 与 vLLM 社区合作集成 vLLM 在Xeon® 服务器的開發和性能优化
  - ✓ RFC: https://github.com/vllm-project/vllm/issues/3654
  - ✓ 文档: https://docs.vllm.ai/en/latest/getting_started/cpu-installation.html
- Intel® Xeon® 服务器是 LLM 推理一个不错的选择

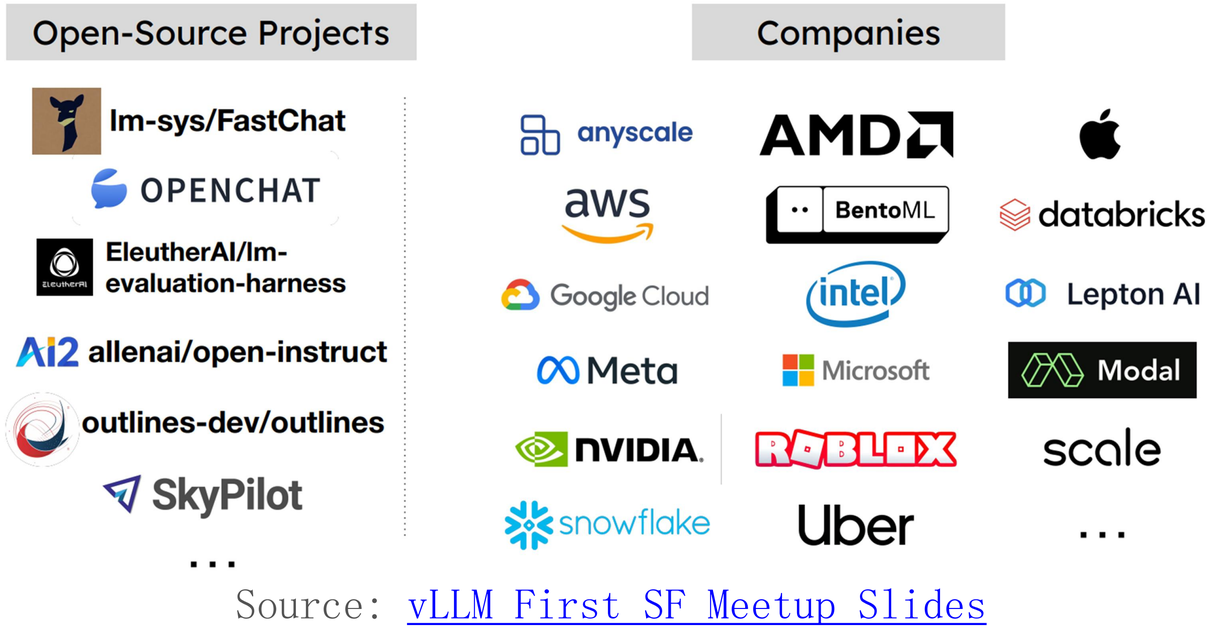
Agenda

- vLLM 简介
- vLLM 在 Intel® Xeon®服务器
- 面向数据中心的Intel® Xeon®服务器
- 案例分析
  - ✓ 案例一：Synchronization
  - ✓ 案例二：Software Prefetching
  - ✓ 案例三： Loop Parallelization

vLLM & CPU backend

vLLM: 使用 Paged Attention 高效管理 LLM 推理服务内存

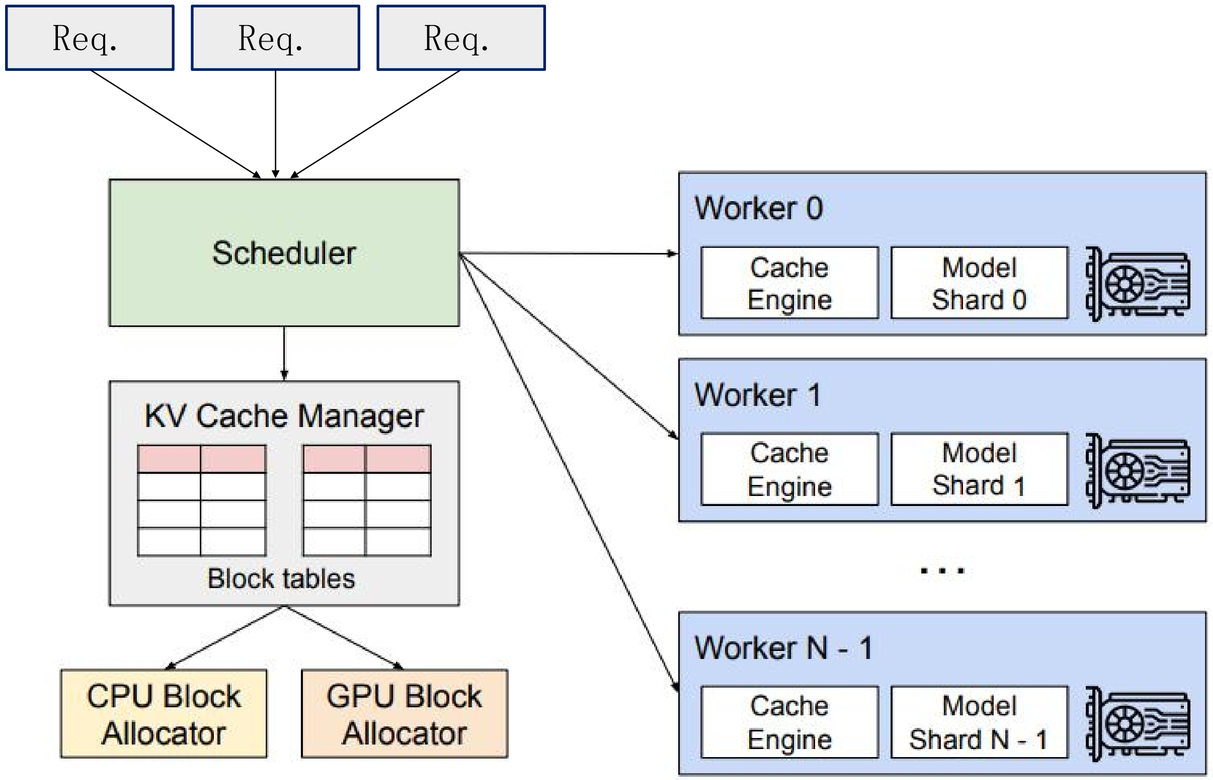
- vLLM 是一套基于*PyTorch* 和 *Huggingface* 的大语言模型推理服务框架
- 由 UC Berkeley Sky Computing Lab 发起
- 关键技术:
  - Continuous batching
  - Paged attention
- Repo: https://github.com/vllm-project/vllm



Source: vLLM First SF Meetup Slides

快速易用的开源 LLM 推理服务引擎

- 高效的*分页 KV Cache* 内存管理
- *Continuous batching*, *tensor parallelism*
- *4-bit 模型量化*, *8-bit KV cache 量化*
- 无缝支持大量 Hugging Face 模型
- 多样的后端设备支持：*NV GPU*, *AMD GPU*, *AWS Inferentia*, *x86 CPU*. (TODO: Intel GPU, Intel Gaudi, Google TPU)



Source: https://arxiv.org/pdf/2309.06180.pdf

快速易用的开源 LLM 推理服务引擎

```python
from vllm import LLM
# Example prompts.
prompts = ["Hello, my name is",
           "The capital of France is"]
# Create an LLM with HF model name.
llm = LLM(model="meta-llama/Llama-2-7b-hf")
# Generate texts from the prompts.
outputs = llm.generate(prompts)
```

离线批处理

Server:

```
$ python -m vllm.entrypoints.openai.api_server
--model meta-llama/Llama-2-7b-hf
```

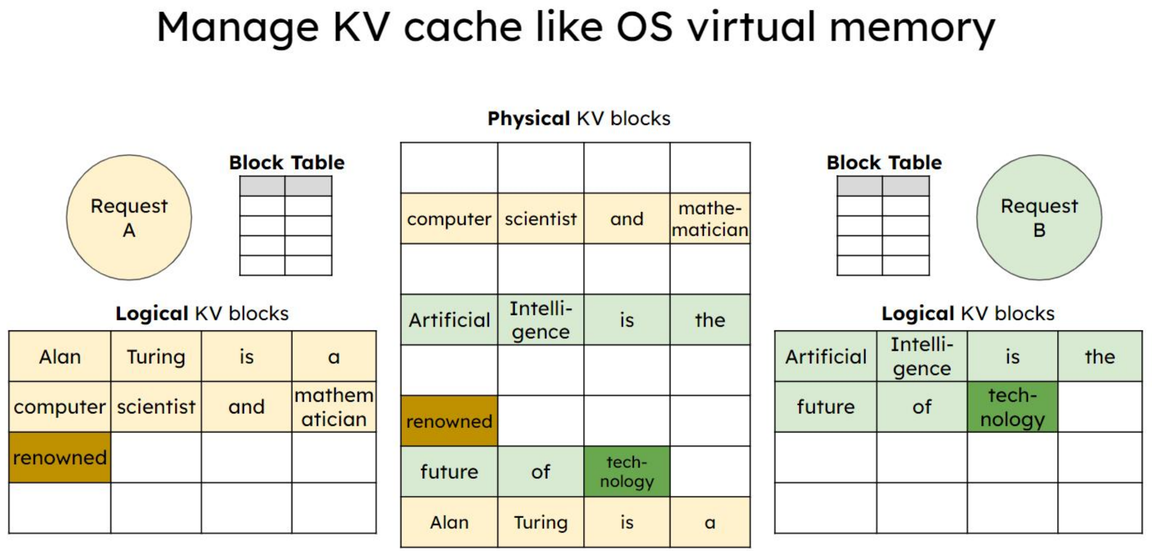
Client:

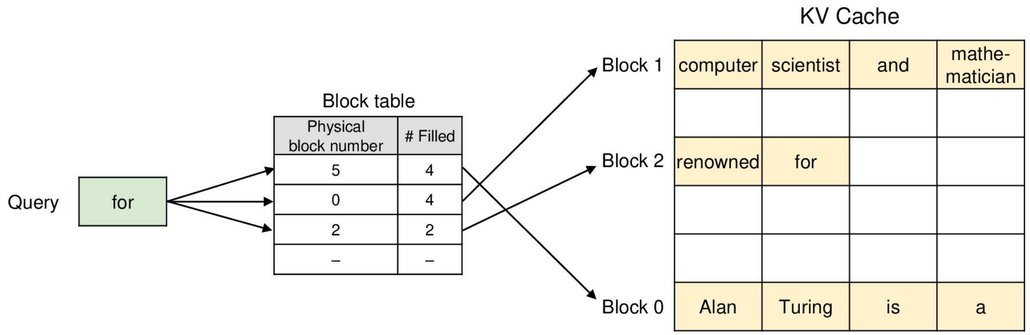
```
$ curl http://localhost:8000/v1/completions \
    -H "Content-Type: application/json" \
    -d '{
        "model": "meta-llama/Llama-2-7b-hf",
        "prompt": "San Francisco is a",
        "max_tokens": 7,
        "temperature": 0
    }'
```

OpenAI-compatible 在线服务

Paged Attention

- LLM 推理服务内存负载特征：
  - **生成序列长度不确定.**
    - 过度预留 KV cache 内存导致浪费
  - **每次迭代动态分配内存.**
    - 内存频繁扩容导致 memcpy
    - 内存碎片
  - **多种特性与算法需要共享 KV cache：**
    - 解码算法 （Beam search, …）
    - 前缀缓存





Source: [vLLM First SF Meetup Slides]

vLLM & Intel

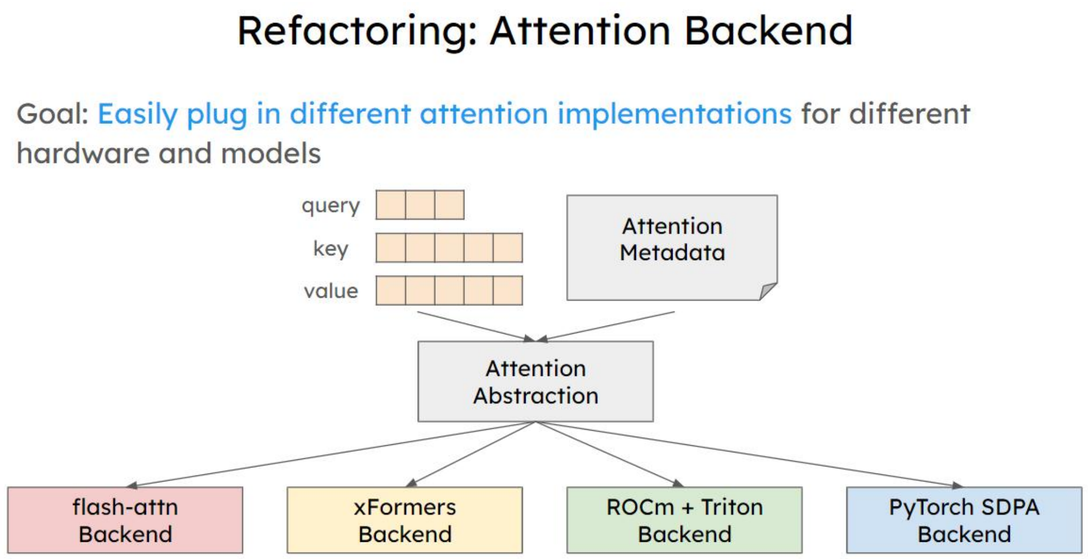
- Intel 与 vLLM 社区合作集成 Intel CPU, GPU 和 Gaudi 后端.
  - **CPU 后端:**
    - RFC: https://github.com/vllm-project/vllm/issues/3654
    - 文档: https://docs.vllm.ai/en/latest/getting started/cpu-installation.html
    - 在 vLLM v0.4.0 合入主分支
  - **GPU 后端:**
    - RFC: https://github.com/vllm-project/vllm/issues/3725
    - 原型: https://github.com/vllm-project/vllm/pull/3814
  - **Gaudi backend:**
    - 原型: https://github.com/HabanaAI/vllm-fork/tree/habana main

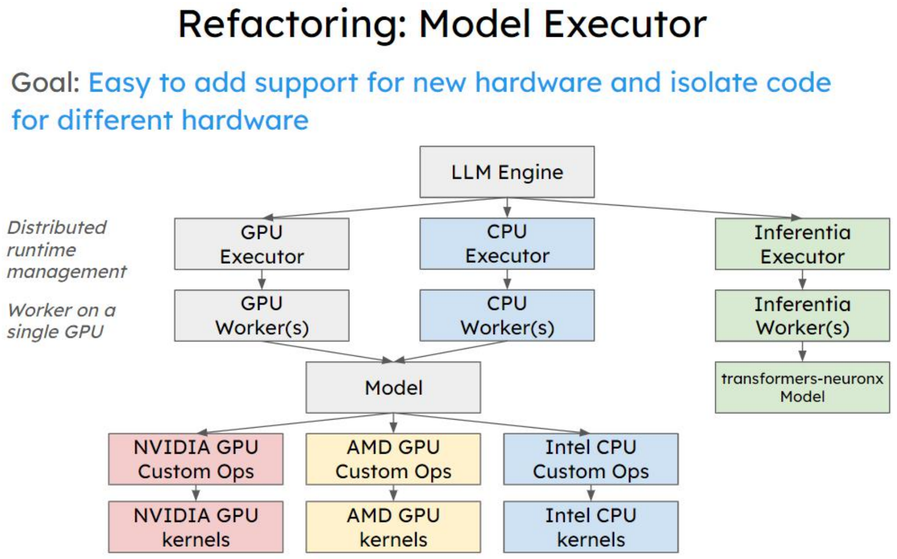
vLLM CPU 后端

- 第四代Xeon处理器（SPR）引入高级矩阵扩展指令集（AMX）来加速矩阵乘法（GEMM）任务:
  - 2D 寄存器和高吞吐专用指令。
  - 支持 INT/UINT8（INT32 累加），FP16，BF16（FP32 累加）.
  - 可编程（汇编或编译器内建函数）
- 最新 cpufp（https://github.com/pigirons/cpufp）加入了对 AMX 的计算性能测试:
  - 使用 AMX，单个 SPR CPU core 在稠密 BF16 GEMM 计算任务上可提供 ~3.7 TFLOPS 的理论计算性能.
  - 相比于 AVX512_BF16（~110 GFLOPS）
  - 入门级数据中心 GPU（~125 TFLOPS）vs. 32 SPR cores（~120 TFLOPS）

vLLM CPU 后端

- GEMM 是大模型推理服务中最关键的负载
- AMX 可通过 oneDNN 数学库或 PyTorch CPU 后端在支持的 CPU 平台上直接使用
- 通过默认的 Pytorch CPU 支持和自定义的高性能 Paged Attention 算子，vLLM CPU 后端可以实现无缝集成





Source: [vLLM Third Meetup Slides]

vLLM CPU 后端

当前状态：

- 支持所有 vLLM 模型中非 MoE 模型在 BF16/FP32 下的推理
- 支持离线批处理

计划功能：

- FP16 支持
- 单机多 Socket Tensor Parallelism
- 4-bit 模型量化 & 8-bit KV cache 量化
- 在线推理服务

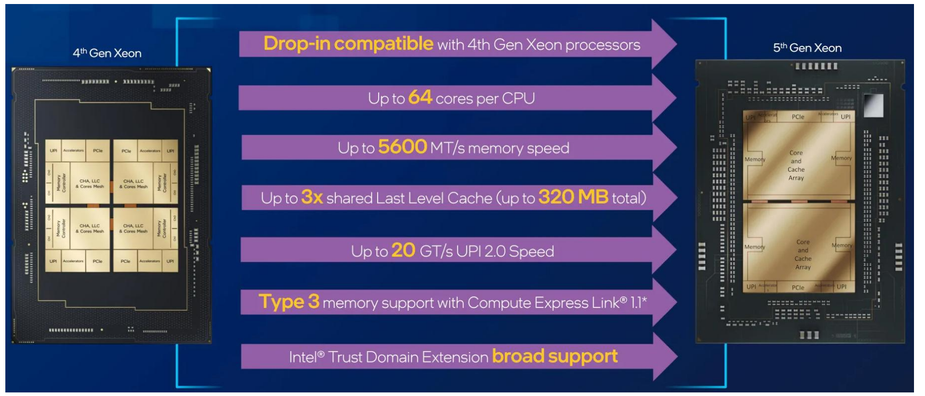
优点：

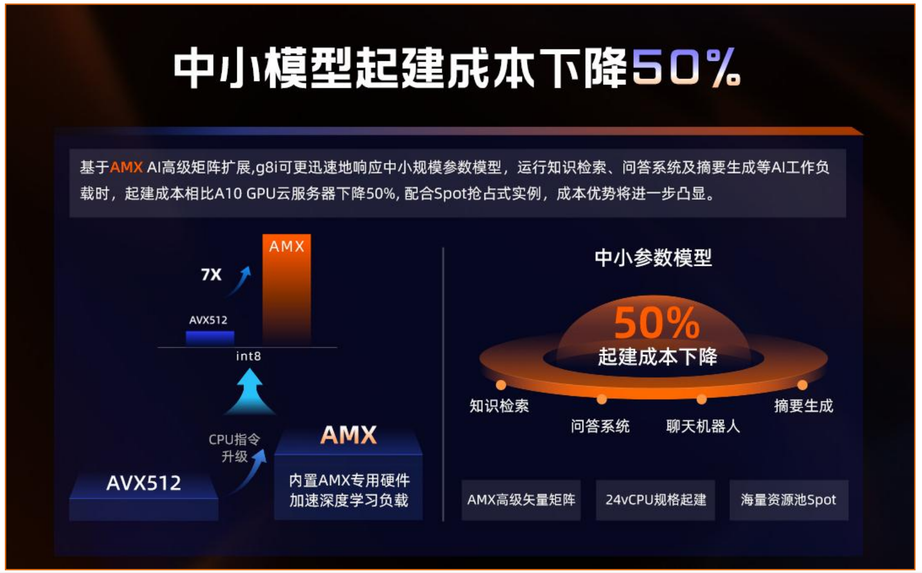
- 在离线批处理任务下，vLLM CPU 后端在 32 核 SPR处理器平台上的吞吐量与一张入门级数据中心 GPU 持平
- 相较于入门级 GPU，CPU 推理可利用更大的 KV cache空间从而支持更长的上下文

Intel® Xeon® 服务器

5th Gen Intel® Xeon® Processors: Designed for AI

- 采用 AI 加速构建
  - AMX (Advanced Matrix Extensions)
  - AVX512
- 更多核、更大缓存、更高内存带宽
- 降低功耗并提高整体效率
  - From a four-tile design (4th generation) to a two-tile die design (5th generation)
- 阿里云在博客中提到TCO降低50% (CPU vs. A10): source

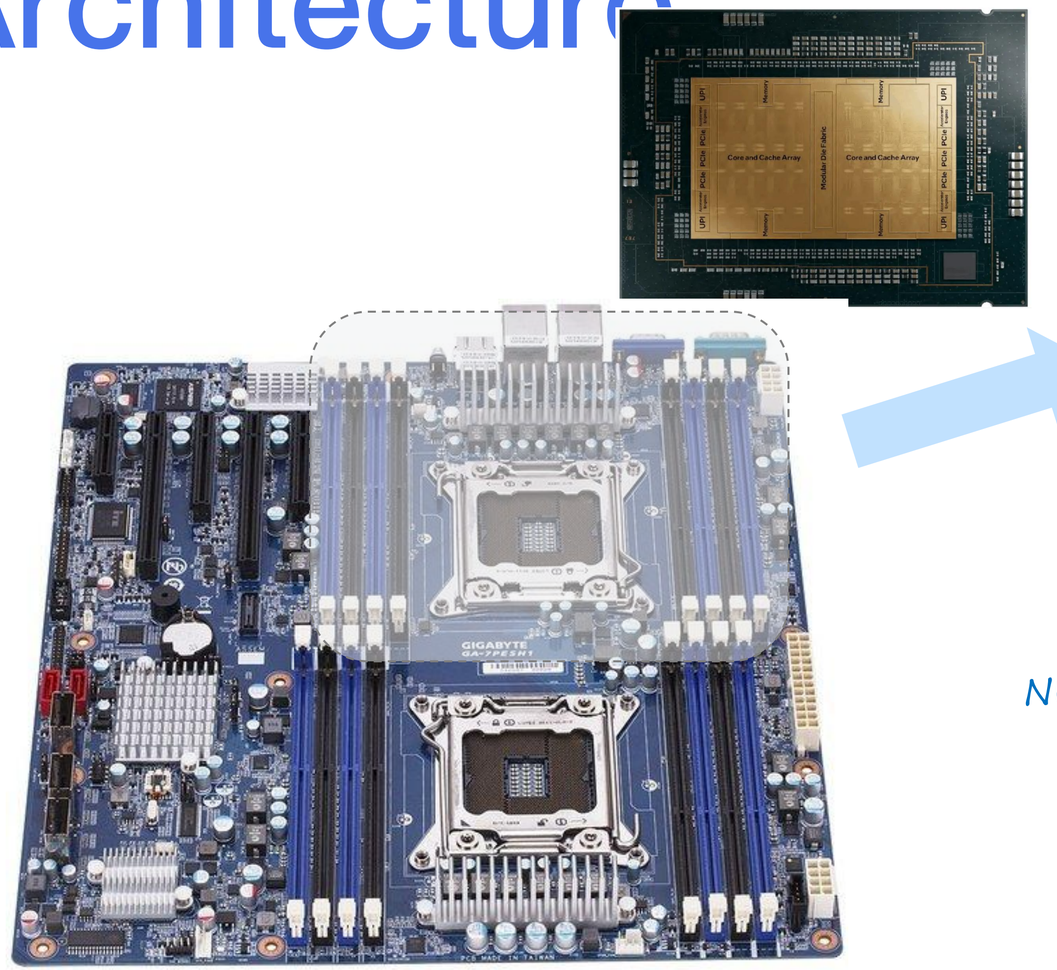


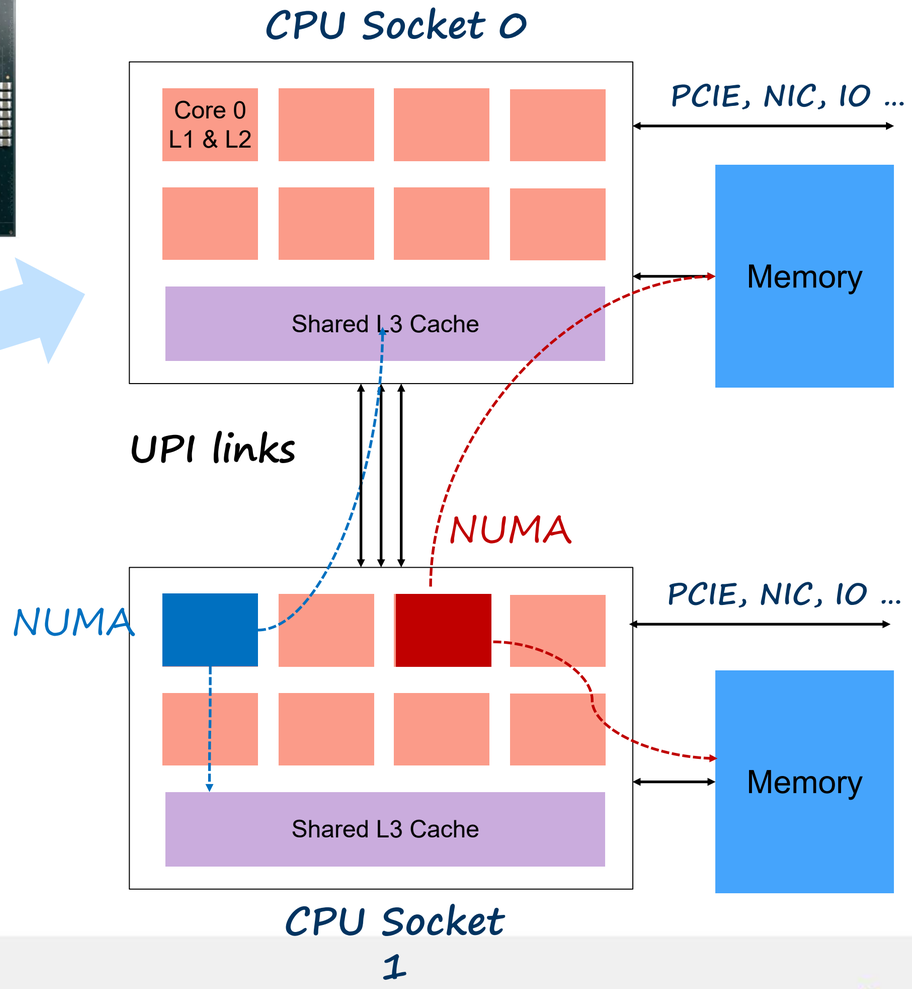


https://cdrdv2.intel.com/v1/dl/getContent/671488

https://github.com/intel/optimization-manual

Intel® Xeon® System: NUMA Architecture





- 案例分析
  - ✓ 案例一：Synchronization
  - ✓ 案例二：Software Prefetching
  - ✓ 案例三： Loop Parallelization

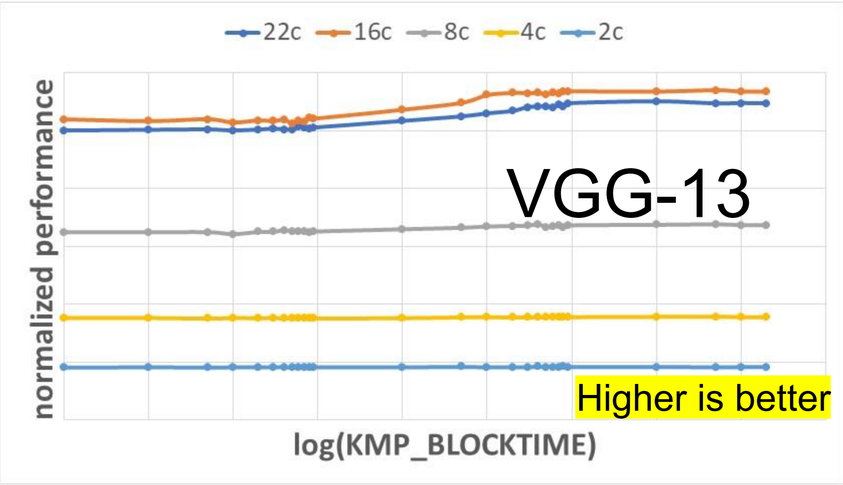
案例一：Synchronization

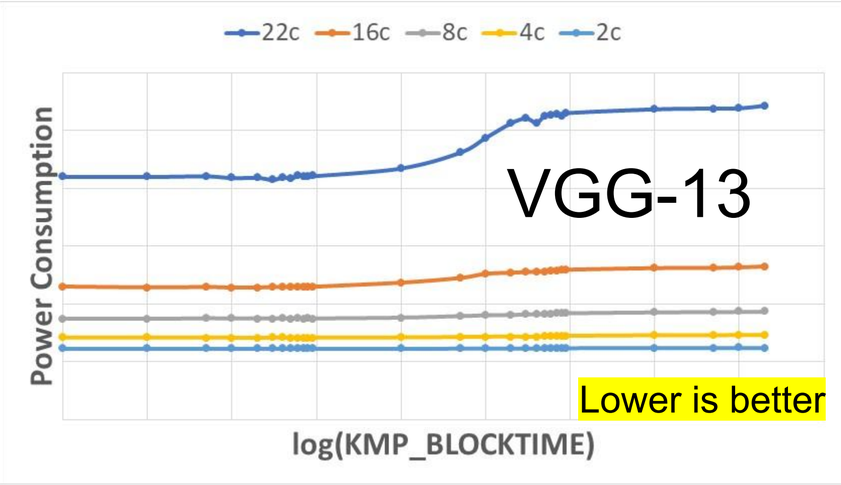
- 多年来，OpenMP 库一直是 AI 深度学习模型的基础
  - OpenMP：一个主线程fork出指定数量的子线程，系统在它们之间划分任务
  - 然后，线程同时运行，运行时环境将线程分配给不同的处理器
  - 保证高并发和高效率一直是一个挑战
- Intel OpenMP vs. GNU OpenMP
  - Intel OpenMP 专门 Intel processors 设计和优化
- Synchronization 线程之间的同步是通过KMP_BLOCKTIME实现的
  - 它设置线程在完成并行区域的执行后、休眠之前应等待的时间

Different heuristics are needed for ransformers (non-CNN topologies) and CNN based models.

Synchronization: 性能和功效

- 許多核服務器：正确设置 KMP_BLOCKTIME 对性能和电能效率都至关重要
  - KMP_BLOCKTIME 值非常高可能会导致 CPU 資源浪费
- 对于少量核服務器，KMP_BLOCKTIME 的设置相對簡單
  - 并行化相对容易

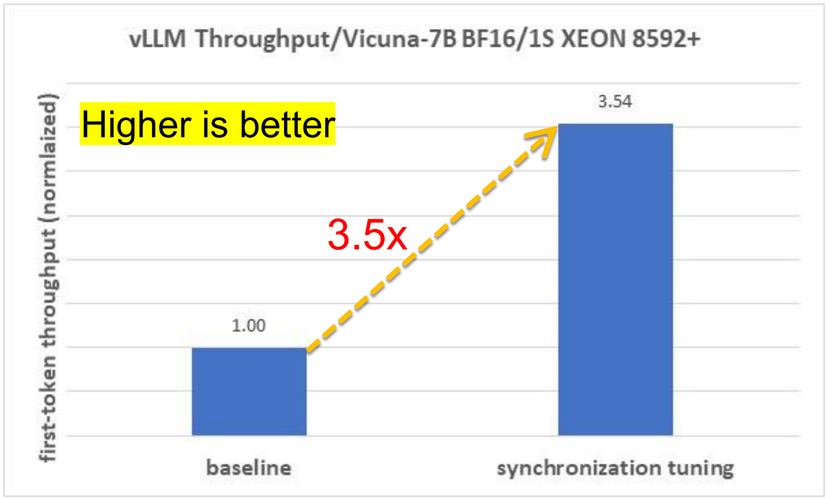


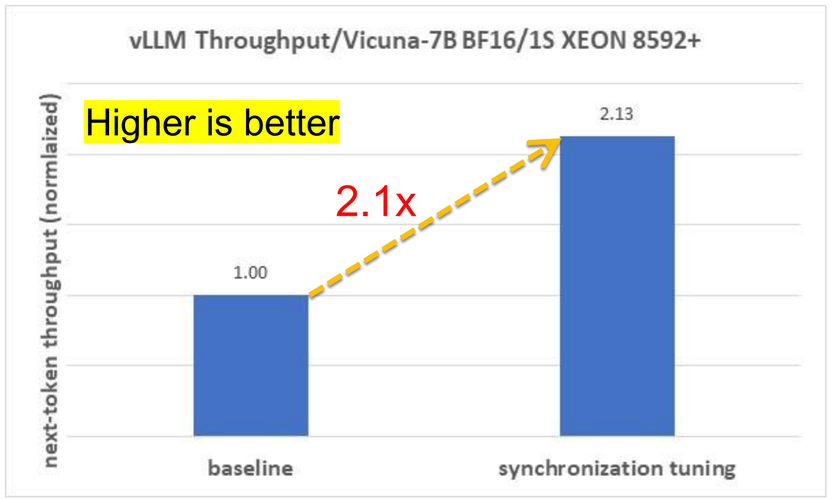


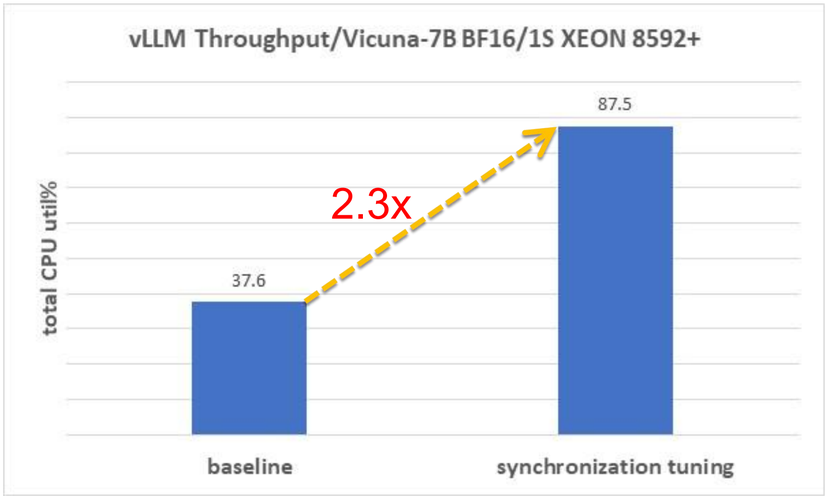
Optimal setting for KMP_BLOCKTIME are workload and system dependent.

Synchronization 的性能影响

- 较高的 KMP_BLOCKTIME（100us→500us）会导致较高的 CPU 利用率（~90%）
  - Spinning 会使用额外的 CPU 資源，但会提高线程使用率
- first-token 和 next-token 吞吐量分别提高3.5x and 2.1x
  - First-token 受益更多，因为它更受 CPU 限制（CPU bound）



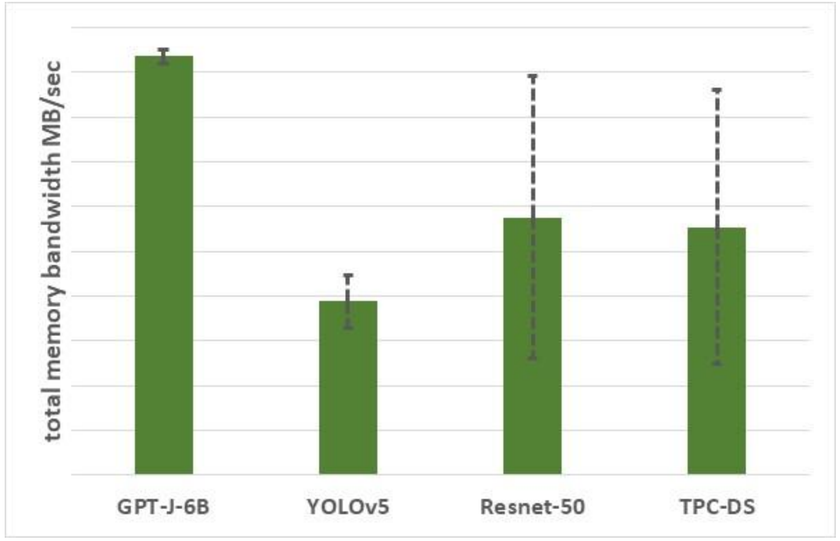


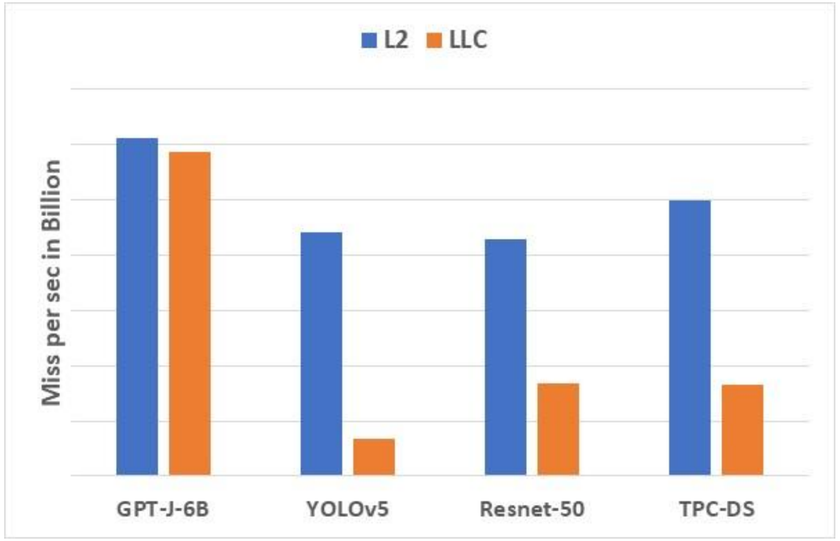


If KMP_BLOCKTIME is not correctly configured, it could result in poor performance.

案例二：Software Prefetch

- LLM Transformers 對内存带宽使用率高
  - 甚至高于一些众所周知的内存带宽密集型應用, e.g., CNN based models (YOLOv5/ResNet-50), Analytics TPC-DS
- 为什么内存占用高？ 频繁的缓存未命中！
  - 缓存未命中通常会触发内存流量
  - Transformers 有更频繁的 L2/LLC 缓存未命中
- 一种解决方案: Software Prefetching
  - 为什么不 Hardware Prefetch?





To reduce cache miss penalty or cache miss rate would alleviate issue.

PagedAttention: VTune Data and Source Code

Function / Call Stack	INST%	CLK%	L2DemandM%	L3DemandM%
(anonymous namespace)::paged_attention_v1_impl<c10::BFloat16, (int)128, (int)16>::call._omp_fn.0	31.3%	44.2%	59.0%	88.0%
[Outside any known module]	24.9%	48.2%	37.9%	11.3%
(anonymous namespace)::fused_add_rms_norm_cpu_impl<c10::BFloat16>._omp_fn.0	1.4%	0.4%	0.3%	0.2%
(anonymous namespace)::rotary_embedding_impl<c10::BFloat16>._omp_fn.0	0.8%	0.2%	0.1%	0.1%

- PagedAttention: the contiguous logical blocks of a sequence are mapped to non-contiguous physical blocks via a block table
  - Hardware Prefetcher难以识别數據的访问模式
- 热函数 <paged_attention_v1_impl> 占总 CPU 使用的 44%

261	**// Compute value [Block 29]**	***51%***
262	constexpr int head_elem_num_per_partition = 16;	
263	constexpr int head_partition_num =	
264	HEAD_SIZE / head_elem_num_per_partition;	
265	for (int head_part_idx = 0; head_part_idx < head_partition_num;	
266	++head_part_idx) {	
267	vec_op::FP32Vec16 accums[head_elem_num_per_partition];	
268	scalar_t *__restrict__ out_ptr =	
269	out + seq_idx * num_heads * HEAD_SIZE + head_idx * HEAD_SIZE +	
270	head_part_idx * head_elem_num_per_partition;	
271	**for (int block_idx = 0; block_idx < block_num; ++block_idx) {**	
272	const int64_t physical_block_idx = seq_block_table[block_idx];	
273	const float *__restrict__ prob_vec_ptr =	
274	thread_block_logits + block_idx * BLOCK_SIZE;	
275	const scalar_t *__restrict__ v_block_cache_ptr =	
276	**v_cache + physical_block_idx * kv_block_stride +**	
277	**kv_head_idx * kv_head_stride +**	
278	BLOCK_SIZE * head_part_idx * head_elem_num_per_partition;	

- [Block 29] 占总 CPU 使用的 [51%*44%]
- Software prefetch 优化的最佳候选者
  - 占全部 total L2 demand miss 的27%
  - HWPF coverage 相对较低

PagedAttention: Hotspot and Assembly Code

Address	Source Line	Assembly	coverage	INST%	CLK%	L2DemandM%	L3DemandM%
0x21120	0	Block 29:	35%				
0x21120	272	movsxdl (%r15,%rdx,4), %rax		0.6%	0.0%	0.0%	0.0%
0x21124	0	mov %rdx, %rdi		0.5%			
0x21127	271	add $0x1, %rdx		0.6%	0.0%	0.0%	0.0%
0x2112b	0	shl $0x6, %rdi		0.6%	0.1%	0.4%	0.0%
0x2112f	276	imul %rcx, %rax		0.6%	0.0%	0.0%	
0x21133	0	vmovupsz (%r12,%rdi,1), %zmm0		0.5%	0.0%	0.0%	0.0%
0x2113a	277	add %rbx, %rax		0.6%	0.0%	0.0%	0.0%
0x2113d	277	lea (%r8,%rax,2), %rax		0.5%	0.1%	0.4%	0.8%
0x21141	0	vpmovsxwdy (%rax), %zmm16		0.5%	9.4%	10.8%	0.0%
0x21147	0	vpslld $0x10, %zmm16, %zmm16		0.5%	0.1%	0.2%	7.6%
0x2114e	0	vfmadd231ps %zmm16, %zmm0, %zmm20		0.5%	0.2%	0.4%	0.1%
0x21154	0	vpmovsxwdy 0x20(%rax), %zmm16		0.5%	0.0%	0.0%	0.1%
0x2115b	0	vpslld $0x10, %zmm16, %zmm16		0.5%	0.1%	0.1%	0.1%
0x21162	0	vfmadd231ps %zmm16, %zmm0, %zmm15		0.5%	0.2%	0.3%	0.7%
0x21168	0	vpmovsxwdy 0x40(%rax), %zmm16		0.6%	5.0%	5.2%	0.2%
0x2116f	0	vpslld $0x10, %zmm16, %zmm16		0.5%	0.1%	0.1%	4.5%
0x21176	0	vfmadd231ps %zmm16, %zmm0, %zmm13		1.0%	0.1%	0.5%	
0x2117c	0	vpmovsxwdy 0x60(%rax), %zmm16		0.5%	0.0%	0.0%	0.1%
0x21183	0	vpslld $0x10, %zmm16, %zmm16		0.6%	0.0%	0.1%	0.0%
0x2118a	0	vfmadd231ps %zmm16, %zmm0, %zmm12		0.5%	0.2%	0.3%	0.5%
0x21190	0	vpmovsxwdy 0x80(%rax), %zmm16		0.5%	5.0%	5.6%	0.4%
0x21197	0	vpslld $0x10, %zmm16, %zmm16		0.5%	0.1%	0.1%	4.3%
0x2119e	0	vfmadd231ps %zmm16, %zmm0, %zmm11		0.5%	0.2%	0.3%	0.1%
0x211a4	0	vpmovsxwdy 0xa0(%rax), %zmm16		0.6%	0.0%	0.0%	0.1%
0x211ab	0	vpslld $0x10, %zmm16, %zmm16		0.6%	0.0%	0.1%	0.0%
0x211b2	0	vfmadd231ps %zmm16, %zmm0, %zmm10		0.5%	0.2%	0.3%	0.3%
0x211b8	0	vpmovsxwdy 0xc0(%rax), %zmm16		0.6%	4.9%	6.0%	0.5%
0x211bf	0	vpslld $0x10, %zmm16, %zmm16		0.6%	0.1%	0.1%	4.1%
0x211c6	0	vfmadd231ps %zmm16, %zmm0, %zmm9		0.6%	0.2%	0.4%	0.1%
0x211cc	0	vpmovsxwdy 0xe0(%rax), %zmm16		0.6%	0.0%	0.0%	0.1%
0x211d3	0	vpslld $0x10, %zmm16, %zmm16		0.6%	0.1%	0.1%	0.0%
0x211da	0	vfmadd231ps %zmm16, %zmm0, %zmm8		0.5%	0.2%	0.4%	0.1%
0x211e0	0	vpmovsxwdy 0x100(%rax), %zmm16		0.6%	4.8%	6.1%	0.6%
0x211e7	0	vpslld $0x10, %zmm16, %zmm16		0.5%	0.1%	0.1%	3.7%
0x211ee	0	vfmadd231ps %zmm16, %zmm0, %zmm7		0.6%	0.2%	0.4%	0.1%
0x211f4	0	vpmovsxwdy 0x120(%rax), %zmm16		0.5%	0.0%	0.0%	0.1%
0x211fb	0	vpslld $0x10, %zmm16, %zmm16		0.5%	0.0%	0.1%	0.0%
0x21202	0	vfmadd231ps %zmm16, %zmm0, %zmm5		0.6%	0.2%	0.5%	0.1%
0x21208	0	vpmovsxwdy 0x140(%rax), %zmm16		0.5%	4.4%	6.0%	0.7%
0x2120f	0	vpslld $0x10, %zmm16, %zmm16		0.6%	0.1%	0.1%	3.3%
0x21216	0	vfmadd231ps %zmm16, %zmm0, %zmm4		0.5%	0.2%	0.5%	0.1%
0x2121c	0	vpmovsxwdy 0x160(%rax), %zmm16		0.6%	0.0%	0.0%	0.1%
0x21223	0	vpslld $0x10, %zmm16, %zmm16		0.5%	0.1%	0.1%	0.0%
0x2122a	0	vfmadd231ps %zmm16, %zmm0, %zmm3		0.6%	0.2%	0.6%	0.1%
0x21230	0	vpmovsxwdy 0x180(%rax), %zmm16		0.5%	4.9%	7.3%	0.7%
0x21237	0	vpslld $0x10, %zmm16, %zmm16		0.6%	0.1%	0.1%	3.4%
0x2123e	0	vfmadd231ps %zmm16, %zmm0, %zmm2		0.6%	0.2%	0.6%	0.1%
0x21244	0	vpmovsxwdy 0x1a0(%rax), %zmm16		0.5%	0.0%	0.0%	0.1%
0x2124b	0	vpslld $0x10, %zmm16, %zmm16		0.6%	0.0%	0.1%	0.0%
0x21252	0	vfmadd231ps %zmm16, %zmm0, %zmm1		0.5%	0.2%	0.6%	0.1%
0x21258	0	vpmovsxwdy 0x1c0(%rax), %zmm16		0.6%	7.9%	11.3%	0.7%
0x2125f	0	vpslld $0x10, %zmm16, %zmm16		0.5%	0.1%	0.1%	4.1%
0x21266	0	vfmadd231ps %zmm16, %zmm0, %zmm19		0.5%	0.2%	0.5%	0.1%
0x2126c	0	vpmovsxwdy 0x1e0(%rax), %zmm16		0.6%	0.0%	0.0%	0.1%
0x21273	0	vpslld $0x10, %zmm16, %zmm16		0.6%	0.1%	0.1%	0.0%
0x2127a	0	vfmadd231ps %zmm16, %zmm0, %zmm18		0.5%	0.2%	0.5%	0.1%
0x21280	271	cmp %edx, %r13d		0.0%			0.8%
0x21283	271	jnle 0x21120 <Block 29>		1.0%	0.1%	0.5%	

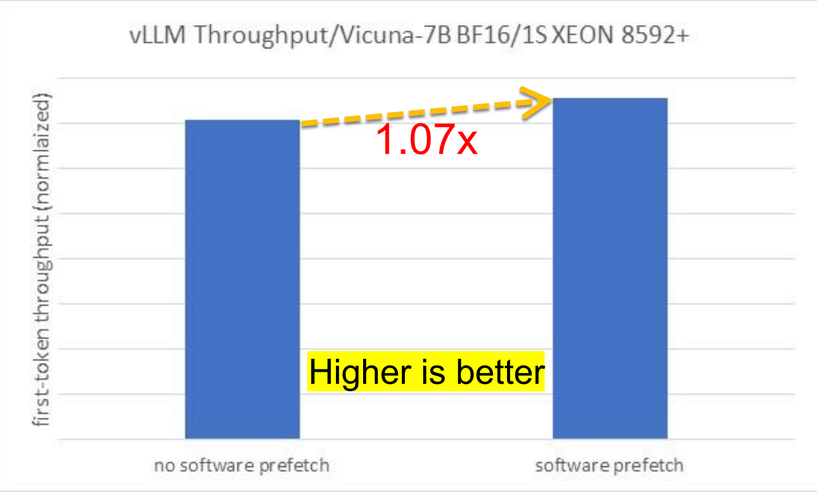
Address	Source Line	Assembly	coverage	INST%	CLK%	L2DemandM%	L3DemandM%
0x21120	0	***Block 29: (sum)***	35%				
0x21120	272	movsxdl (%r15,%rdx,4), %rax		0.6%	0.0%	0.0%	0.0%
0x21124	0	mov %rdx, %rdi		0.5%			
0x21127	271	add $0x1, %rdx		0.6%	0.0%	0.0%	0.0%
0x2112b	0	shl $0x6, %rdi		0.6%	0.1%	0.4%	0.0%
0x2112f	276	imul %rcx, %rax		0.6%	0.0%	0.0%	
0x21133	0	vmovupsz (%r12,%rdi,1), %zmm0		0.5%	0.0%	0.0%	0.0%
0x2113a	277	add %rbx, %rax		0.6%	0.0%	0.0%	0.0%
0x2113d	277	lea (%r8,%rax,2), %rax		0.5%	0.1%	0.4%	0.8%
0x21141	0	vpmovsxwdy (%rax), %zmm16		0.5%	9.4%	10.8%	0.0%
0x21147	0	vpslld $0x10, %zmm16, %zmm16		0.5%	0.1%	0.2%	7.6%
0x2114e	0	vfmadd231ps %zmm16, %zmm0, %zmm20		0.5%	0.2%	0.4%	0.1%
0x21154	0	vpmovsxwdy 0x20(%rax), %zmm16		0.5%	0.0%	0.0%	0.1%
0x2115b	0	vpslld $0x10, %zmm16, %zmm16		0.5%	0.1%	0.1%	0.1%
0x21162	0	vfmadd231ps %zmm16, %zmm0, %zmm15		0.5%	0.2%	0.3%	0.7%
0x21168	0	vpmovsxwdy 0x40(%rax), %zmm16		0.6%	5.0%	5.2%	0.2%
0x2116f	0	vpslld $0x10, %zmm16, %zmm16		0.5%	0.1%	0.1%	4.5%
0x21176	0	vfmadd231ps %zmm16, %zmm0, %zmm13		1.0%	0.1%	0.5%	
0x2117c	0	vpmovsxwdy 0x60(%rax), %zmm16		0.5%	0.0%	0.0%	0.1%
0x21183	0	vpslld $0x10, %zmm16, %zmm16		0.6%	0.0%	0.1%	0.0%
0x2118a	0	vfmadd231ps %zmm16, %zmm0, %zmm12		0.5%	0.2%	0.3%	0.5%
0x21190	0	vpmovsxwdy 0x80(%rax), %zmm16		0.5%	5.0%	5.6%	0.4%

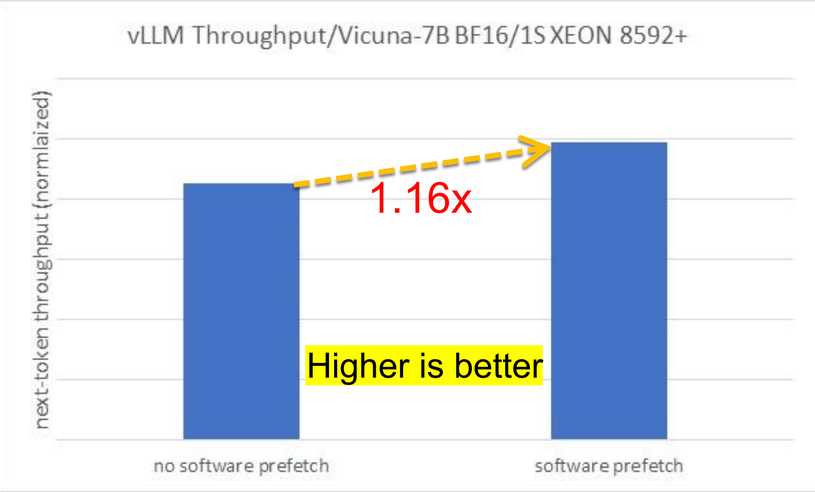
- “Yellow”: major contributor to the misses; “Green”: most are cache hit
- “Yellow” also serves as “prefetch” for the Green (due to the cacheline granularity of 64Bytes).

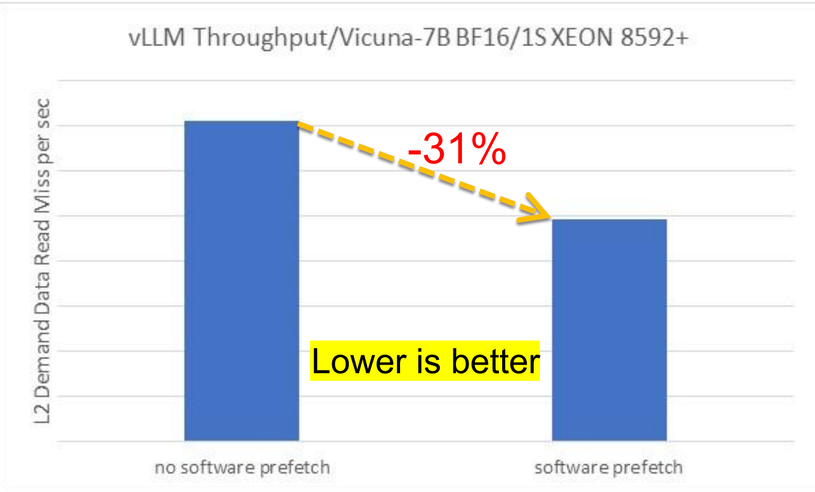
- 占這個熱點函數总 CPU 使用的 51%
- 占這個熱點函數 total L2 demand miss 的~69%
- 占這個熱點函數 total L3 demand miss 的~44%
- HWPF coverage 相對低 (~35%)

Software Prefetching 的性能影响

- first-token 和 next-token 吞吐量分别提高7% and 16%
  - L2_Demand_Data_Read_miss 减少 31%
  - Next-token 性能对缓存未命中/内存带宽更敏感





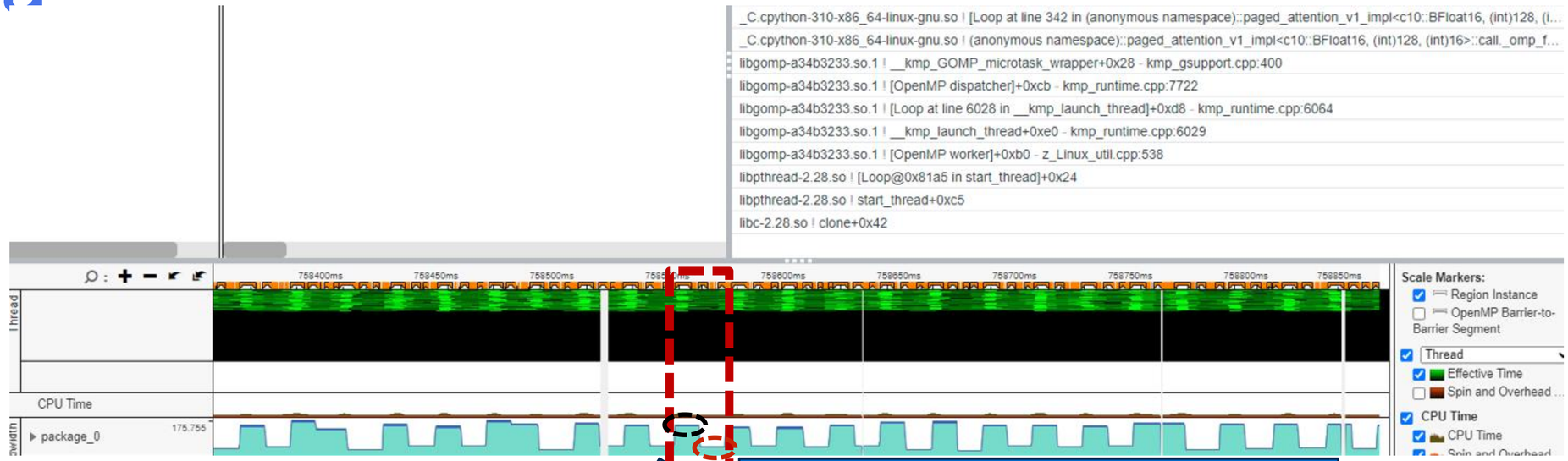


Software prefetching reduces demand data read miss therefore improves throughput.

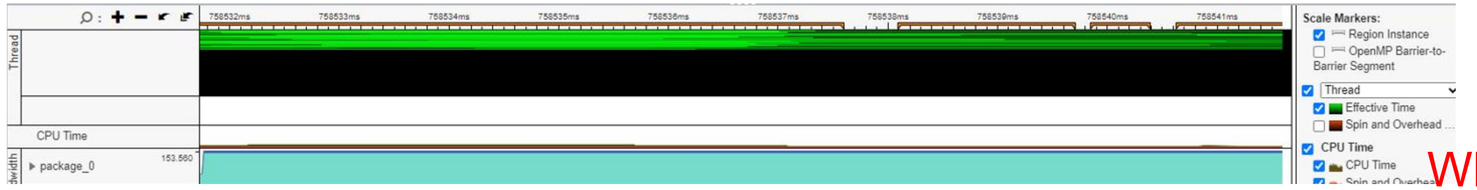
案例三： Loop Parallelization

- 并行化（Parallelization）可以显著提高计算密集型循环（computationally intensive loops）的性能
  - 指示编译器在可用线程之间分配循环的迭代
  - 每个线程执行循环迭代的指定部分
- 性能方面的考量:
  - Parallel region 并行区域大小合理
    - 创建和管理线程的线程开销
  - False Sharing: 数据对齐或填充策略以减少 False Sharing
  - Load imbalance：当某些线程比其他线程更快地完成分配的迭代时，就会出现负载不平衡，从而导致线程空闲，而其他线程仍在工作 ➔ 这降低了并行化parallelization的整体效率
    - Schedule with options [static/dynamic/guided]
- VTune 有一个特殊而强大的功能来识别 Intel OpenMP 性能瓶颈
  - 不兼容GNU OpenMP

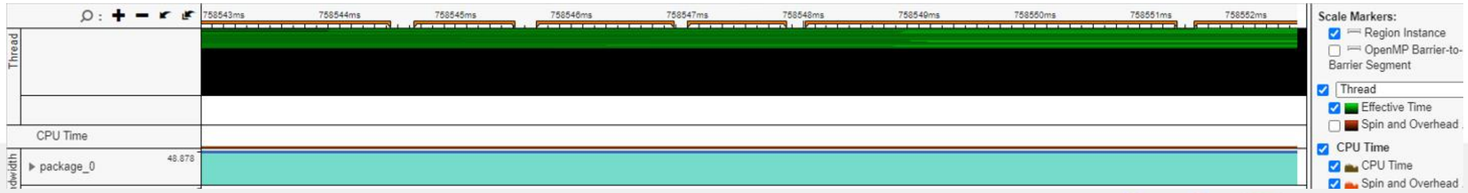
Loop Parallelization: Top–Down with VTune Data



Why memory bandwidth usage is high?



Why memory bandwidth usage is low?



Loop Parallelization 优化

Baseline:

```
struct paged_attention_v1_impl {

    for (int seq_idx = 0; seq_idx < num_seqs; ++seq_idx) {

        // Compute attention logits
#pragma omp parallel for collapse(2)
        for (int block_idx = 0; block_idx < block_num; ++block_idx) {
          for (int head_idx = 0; head_idx < num_heads; ++head_idx) {
...
...
          }
        }

        // Compute softmax
#pragma omp parallel for
        for (int head_idx = 0; head_idx < num_heads; ++head_idx) {
          ...
          ...
          ...
        }

        // Compute value
#pragma omp parallel for collapse(2)
        for (int head_idx = 0; head_idx < num_heads; ++head_idx) {
          for (int head_part_idx = 0; head_part_idx < head_partition_num;
               ++head_part_idx) {
            for (int block_idx = 0; block_idx < block_num; ++block_idx) {
              ...
              ...
              ...

            }
          }
        }

    }
}
```

In each iteration step of the “seq_idx” loop, three parallel regions are created. This causes parallelization overhead

Parallel Optimized:

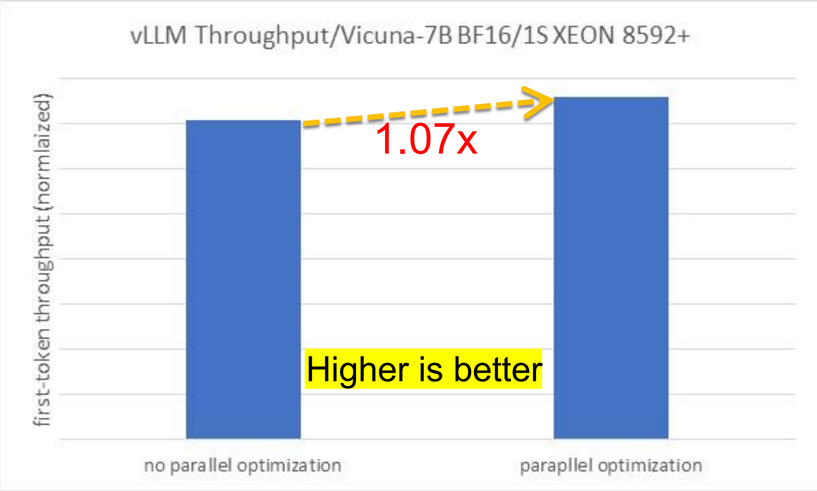
```
struct paged_attention_v1_impl {
    int max_block_num=(max_context_len_padded+BLOCK_SIZE-1)/BLOCK_SIZE;

#pragma omp parallel for collapse(3) schedule(dynamic)
    for (int seq_idx = 0; seq_idx < num_seqs; ++seq_idx) {
      for (int head_idx = 0; head_idx < num_heads; ++head_idx) {
        for (int block_idx = 0; block_idx<max_block_num; ++block_idx) {
...
...
        }
      }
    }

    // Compute softmax
#pragma omp parallel for collapse(2) schedule(dynamic)
    for (int seq_idx = 0; seq_idx < num_seqs; ++seq_idx) {
      for (int head_idx = 0; head_idx < num_heads; ++head_idx) {
...
...
      }
    }

    // Compute value
#pragma omp parallel for collapse(3) schedule(dynamic)
    for (int seq_idx = 0; seq_idx < num_seqs; ++seq_idx) {
      for (int head_idx = 0; head_idx < num_heads; ++head_idx) {
        for (int head_part_idx = 0; head_part_idx < head_partition_num;
             ++head_part_idx) {
          int context_len = context_lens[seq_idx];
...
...
          }
        }
      }
    }
}
```

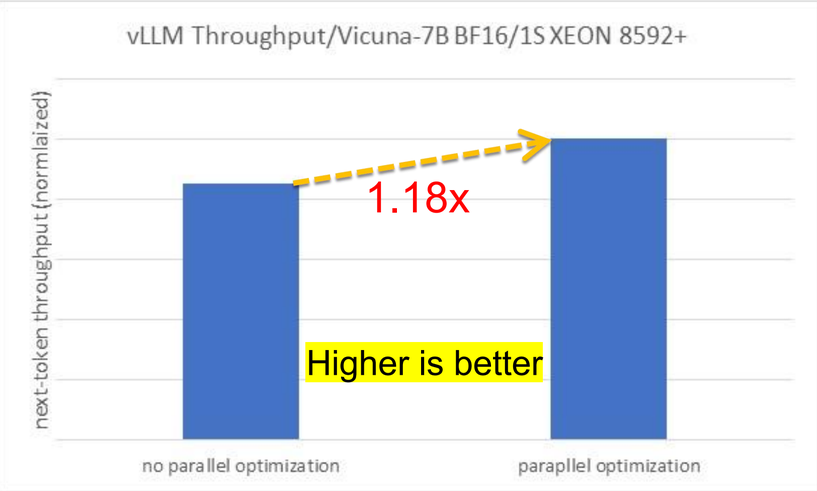
Long parallel region

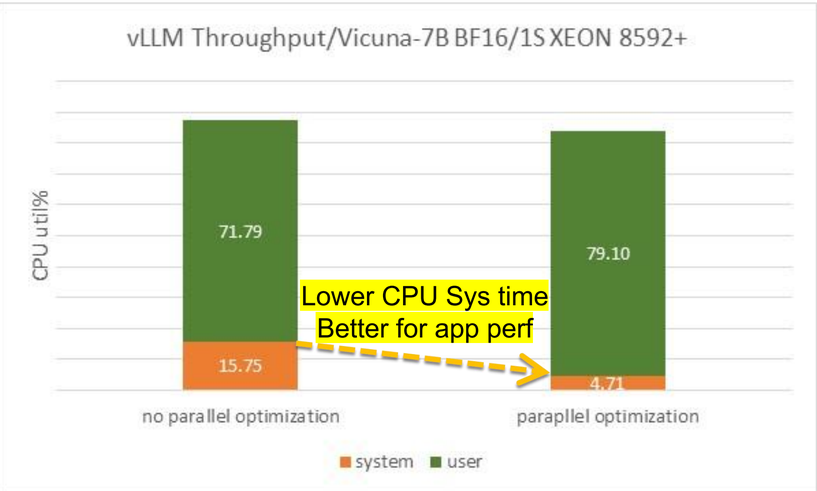
Profiling shows that both parallel regions could be too short. WIP.

Loop Parallelization 的性能影响

- first-token 和 next-token 吞吐量分别提高 7% and 18%
  - 显着减少CPU系统时间
- 给二级缓存 L2 cache 带来额外压力
  - 由于代码变化，以前的 software prefetch 无法直接应用
  - 将应用类似的 software prefetch 方法







© 2024 Intel Corporation

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/performance.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

InfoQ 极客传媒

已结束 北京站 2000 人

QCon 全球软件开发大会 暨智能软件开发生态展

架构演进 | AIGC
研发效能 | 向量数据库

6 月 14–15 日 深圳站 1000 人

ArchSummit 全球架构师峰会 暨智能软件开发生态展

业务架构 | 高可用
AI 架构 | 架构师成长

10 月 18–19 日 上海站 1500 人

QCon 全球软件开发大会 暨智能软件开发生态展

AIGC | AI 原生应用
软件工程 | 金融科技

4月 5月 6月 8月 10月 12月

5 月 17–18 日 北京站 2000 人

AiCon 全球人工智能开发与应用大会 暨大模型应用生态展

通用大模型 | AI Agent
LLMOps | AI Infra

8 月 16–17 日 上海站 1000 人

FCon 全球金融科技大会 暨智能软件开发生态展

金融大模型 | 核心系统
数字化转型 | 数据安全

12 月 20–21 日 北京站 1000 人

ArchSummit 全球架构师峰会 暨智能软件开发生态展

未来架构 | 高可用高并发
AI 应用 | 架构师思维

极客邦科技 2024 年会议规划

促进软件开发及相关领域知识与创新的传播

访问大会官网

参会咨询

THANKS

大模型正在重新定义软件

Large Language Model Is Redefining The Software