目　录

在蝰蛇峡谷上实现 YOLOv5 模型的 OpenVINO 异步推理程序

本文将介绍通过异步推理实现方式，进一步提升 AI 推理程序的性能。在阅读本文前，请读者先参考《基于 OpenVINO™2022.2 和蝰蛇峡谷优化并部署 YOLOv5 模型》，完成 OpenVINO 开发环境的创建并获得 yolov5s.xml 模型，然后阅读范例程序 yolov5_ov2022_sync_dGPU.py，了解了 OpenVINO™ 的同步推理程序实现方式。

1.1 AI 推理程序性能评价指标

在提升 AI 推理程序的性能前，先要理解评估 AI 推理程序性能的指标是什么。我们常用时延(Latency)和吞吐量(Throughput)来衡量 AI 推理程序的性能。

- 时延：测量处理一个单位数据的速度快不快
- 吞吐量：测量一个单位时间里面处理的数据多不多

很多人都容易误认为时延低必然吞吐量高，时延高必然吞吐量低。其实不是这样，以 ATM 机取钱为例，假设一个人在 ATM 机取钱的速度是 30 秒，若 A 银行有两台 ATM 机，那么 A 银行的吞吐量为 4 人/分钟，时延是 30 秒；若 B 银行有 4 台 ATM 机，那么 B 银行的吞吐量为 8 人/分钟，时延是 30 秒；若 C 银行有 4 台 ATM 机，并且要求每个人取完钱后，必须填写满意度调查表，大约花费 30 秒，那么 C 银行的吞吐量为 8 人/分钟，时延为 1 分钟。

由此可见，时延评估的是单一事件的处理速度，吞吐量评估的是整个系统处理事件的效率。时延高低跟吞吐量大小有关系，但不是直接的线性关系，我们需要同时着眼于时延和吞吐量这两个指标去优化。

另外，AI 推理性能评价还有两个常见的场景，一个是单纯评价 AI 模型的推理性能，另一个是整体评价从采集数据到拿到最终结果的端到端的 AI 推理程序性能。

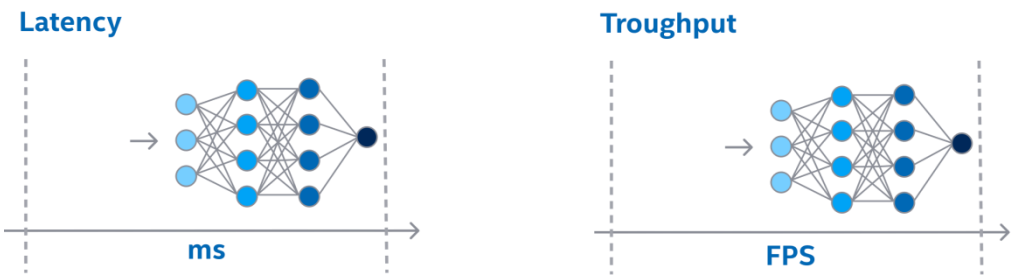
1.1.1 AI 模型的推理性能

在单纯评价 AI 模型的推理性能的场景中：

- 时延具体指：将数据输入 AI 模型后，多长时间可以从 AI 模型拿到输出结果
- 吞吐量具体指：在单位时间能完成多少数据的 AI 推理计算

[**注意**]在单纯评价 AI 模型的推理性能的场景中，数据的前处理和后处理所花费的时间不包含在时延和吞吐量的计算里面。

具体到计算机视觉应用场景的 AI 模型的推理计算性能，吞吐量可以用单位时间内能完成多少张图片的 AI 推理计算来衡量，即 FPS(Frame Per Second)，如下图所示。

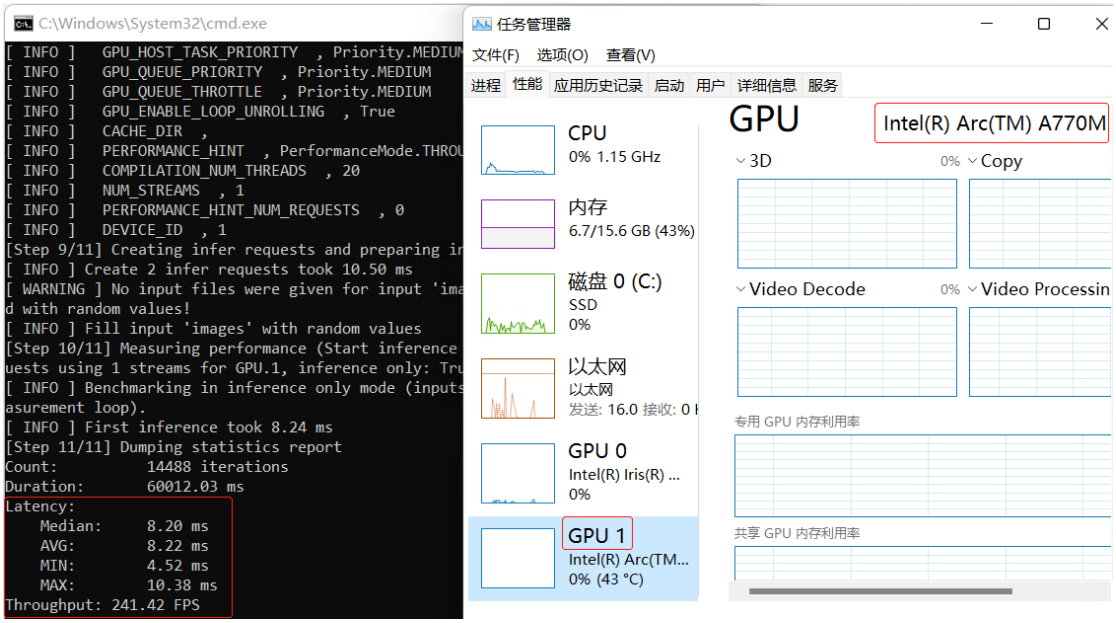


$$\text{FPS} = \frac{\text{inferred number of images}}{\text{processing time in seconds}}$$

OpenVINO 自带的性能评测工具的 benchmark_app，主要用于单纯评价 AI 模型推理性能的场景。在蝰蛇峡谷平台上，使用命令：

```
benchmark_app -m yolov5.xml -d GPU.1
```

可以获得 yolov5.xml 模型在英特尔 A770M 独立显卡(GPU.1)上的推理性能，如下图所示。



1.1.2 端到端的 AI 程序推理性能

当 AI 模型集成到应用程序中后，对用户来说，更加关注的是从采集图像数据到拿到最终结果的端到端的性能，例如，用手机拍了一个水果，用户更在乎的是需要多少时间能展示出这个水果是什么。

一个典型端到端 AI 推理计算程序流程：

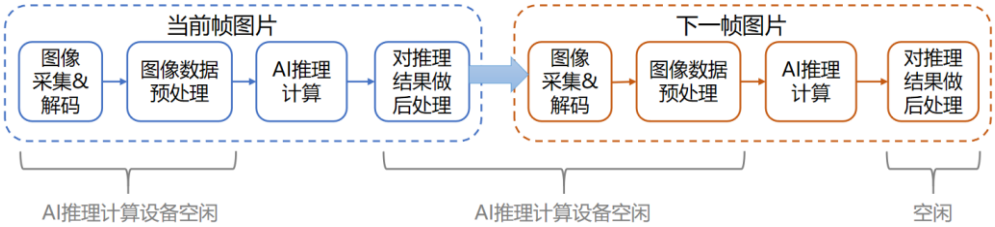
1. 采集图像并解码
2. 根据 AI 模型的要求，对图像数据做预处理
3. 将预处理后的数据送入模型，执行推理计算
4. 对推理计算结果做后处理，拿到最终结果

参考 yolov5_ov2022_sync_dGPU.py 的代码片段

```
# Acquire or load image
```

```
frame = cv2.imread("./data/images/zidane.jpg")
# preprocess frame by letterbox
letterbox_img, _, _= letterbox(frame, auto=False)
# Normalization + Swap RB + Layout from HWC to NCHW
blob = cv2.dnn.blobFromImage(letterbox_img, 1/255.0, swapRB=True)
# Step 3: Do the inference
outs = torch.tensor(net([blob])[output_node])
# Postprocess of YOLOv5:NMS
dets = non_max_suppression(outs)[0].numpy()
bboxes, scores, class_ids= dets[:,:4], dets[:,4], dets[:,5]
# rescale the coordinates
bboxes = scale_coords(letterbox_img.shape[:-1], bboxes, frame.shape[:-1]).astype(int)
```

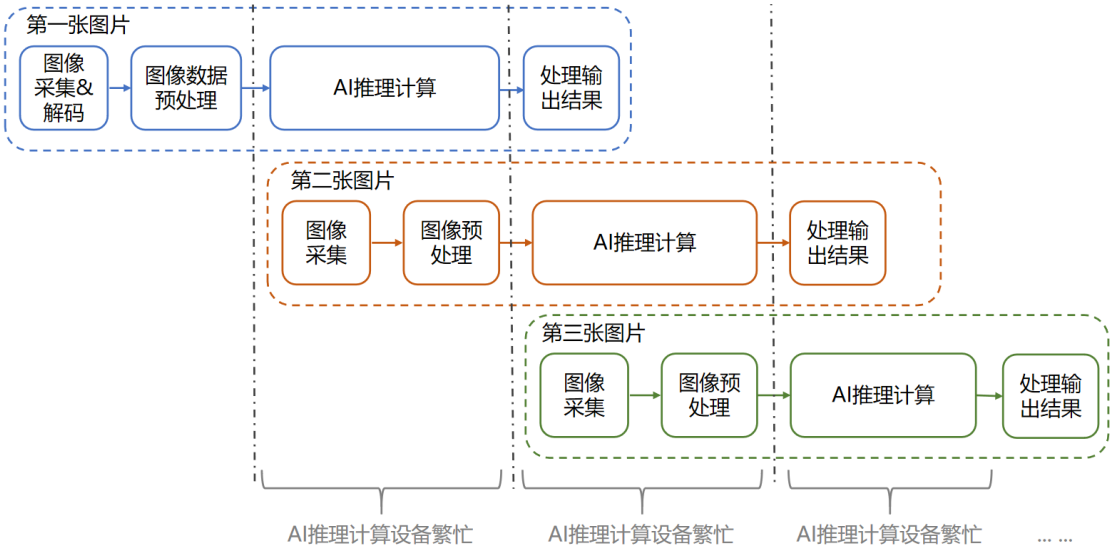
采用 yolov5_ov2022_sync_dGPU.py 中同步实现方式，可以看到在第 1,2,4 步时，AI 推理设备是空闲的，如下图所示：



若能提升 AI 推理设备的利用率，则可以提高 AI 程序的吞吐量。**提升 AI 推理设备利用率的典型方式，是将同步推理实现方式更换为异步推理实现方式**。

1.2 异步推理实现方式

异步推理实现方式是指在当前帧图片做 AI 推理计算时，并行启动下一帧图片的图像采集和图像数据预处理工作，使得当前帧的 AI 推理计算结束后，AI 计算设备可以不用等待，直接做下一帧的 AI 推理计算，持续保持 AI 计算设备繁忙，如下图所示：



使用 benchmark_app 工具，并指定实现方式为同步(sync)或异步(async)，观察性能测试结果，异步方式的确能提高吞吐量，如下图所示：

```
benchmark_app -m yolov5.xml -d GPU.1 -api sync

benchmark_app -m yolov5.xml -d GPU.1 -api async
```

```
[Step 10/11] Measuring performance (Start inference synchronously, in
ference only: True, limits: 60000 ms duration)
[ INFO ] Benchmarking in inference only mode (inputs filling are not
included in measurement loop).
[ INFO ] First inference took 6.16 ms
[Step 11/11] Dumping statistics report
Count:          9874 iterations
Duration:       60003.12 ms
Latency:
    Median:     4.25 ms
    AVG:        4.26 ms
    MIN:        4.08 ms
    MAX:        6.46 ms
Throughput: 235.03 FPS
```

同步方式

```
[Step 10/11] Measuring performance (Start inference asynchronously, 2
 inference requests using 1 streams for GPU.1, inference only: True,
limits: 60000 ms duration)
[ INFO ] Benchmarking in inference only mode (inputs filling are not
included in measurement loop).
[ INFO ] First inference took 6.21 ms
[Step 11/11] Dumping statistics report
Count:          14564 iterations
Duration:       60015.17 ms
Latency:
    Median:     8.16 ms
    AVG:        8.17 ms
    MIN:        4.48 ms
    MAX:        9.96 ms
Throughput: 242.67 FPS
```

异步方式

1.2.1 OpenVINO 异步推理 Python API

OpenVINO™ Runtime 提供了推理请求(Infer Request)机制，来实现在指定的推理设备上以同步或异步方式运行 AI 模型。

在 openvino.runtime.CompiledModel 类里面，定义了 create_infer_request()方法，用于创建 openvino.runtime.InferRequest 对象。

```
infer_request = compiled_model.create_infer_request()
```

当 infer_request 对象创建好后，可以用：

- set_tensor(input_node, input_tensor)：将数据传入模型的指定输入节点
- start_async()：通过**非阻塞(non-blocking)**的方式启动推理计算。
- wait()：等待推理计算结束
- get_tensor(output_node)：从模型的指定输出节点获取推理结果

1.2.2 同步和异步实现方式对比

同步实现方式伪代码	异步实现方式伪代码
创建一个负责处理当前帧的推理请求即可 … … While True: 　采集当前帧图像 　对当前帧做预处理 　**调用 infer()，以阻塞方式启动推理计算** 　对推理结果做后处理 　显示推理结果	创建一个推理请求负责处理当前帧 创建一个推理请求负责处理下一帧 … … 采集当前帧图像 对当前帧做预处理 **调用 start_async()，以非阻塞方式启动当前帧推理计算** While True: 　采集下一帧 　对下一帧做预处理 　**调用 start_async()，以非阻塞方式启动下一帧推理计算** 　**调用 wait()，等待当前帧推理计算结束** 　对当前帧推理结果做后处理 　**交换当前帧推理请求和下一帧推理请求**

1.2.3 异步推理范例程序

根据异步实现方式伪代码，YOLOv5 的异步推理范例程序的核心实现部分如下所示：

完整范例代码请下载：yolov5_ov2022_async_dGPU.py，

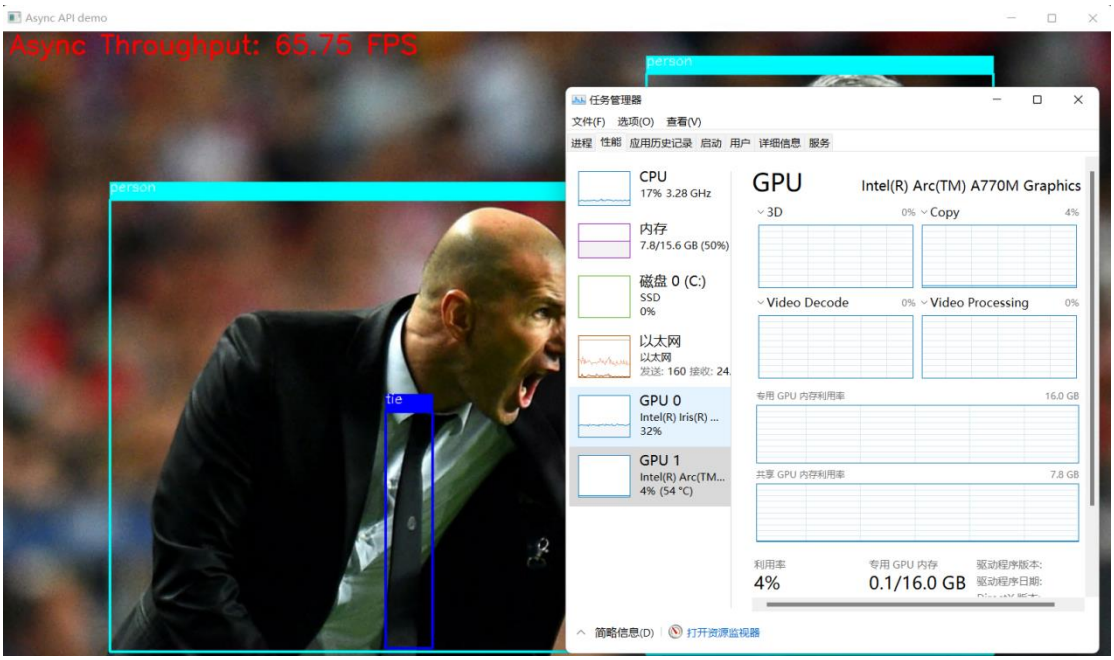
```
...
# Step 3. Create 1 Infer_request for current frame, 1 for next frame
# 创建一个推理请求负责处理当前帧
infer_request_curr = net.create_infer_request()
# 创建一个推理请求负责处理下一帧
infer_request_next = net.create_infer_request()
...
# Get the current frame, 采集当前帧图像
frame_curr = cv2.imread("./data/images/bus.jpg")
```

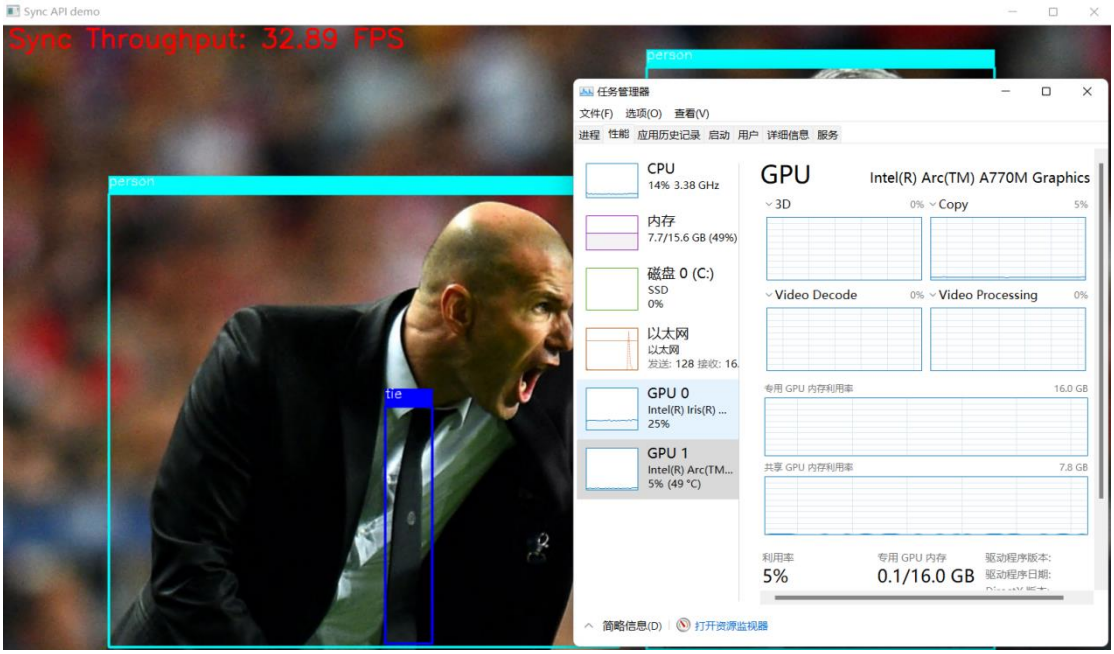
```
# Preprocess the frame，对当前帧做预处理
letterbox_img_curr, _, _ = letterbox(frame_curr, auto=False)
# Normalization + Swap RB + Layout from HWC to NCHW
blob = Tensor(cv2.dnn.blobFromImage(letterbox_img_curr, 1/255.0, swapRB=True))
# Transfer the blob into the model
infer_request_curr.set_tensor(input_node, blob)
# Start the current frame Async Inference，调用start_sync(),以非阻塞方式启动当前帧推理计算
infer_request_curr.start_async()
while True:
    # Calculate the end-to-end process throughput.
    start = time.time()
    # Get the next frame，采集下一帧
    frame_next = cv2.imread("./data/images/zidane.jpg")
    # Preprocess the frame，对下一帧做预处理
    letterbox_img_next, _, _ = letterbox(frame_next, auto=False)
    # Normalization + Swap RB + Layout from HWC to NCHW
    blob = Tensor(cv2.dnn.blobFromImage(letterbox_img_next, 1/255.0, swapRB=True))
    # Transfer the blob into the model
    infer_request_next.set_tensor(input_node, blob)
    # Start the next frame Async Inference，调用start_sync(),以非阻塞的方式启动下一帧推理计算
    infer_request_next.start_async()
    # wait for the current frame inference result，调用wait(),等待当前帧推理计算结束
    infer_request_curr.wait()
    # Get the inference result from the output_node
    infer_result = infer_request_curr.get_tensor(output_node)
    # Postprocess the inference result，对当前帧推理结果做后处理
    data = torch.tensor(infer_result.data)
    # Postprocess of YOLOv5:NMS
    dets = non_max_suppression(data)[0].numpy()
    bboxes, scores, class_ids= dets[:,:4], dets[:,4], dets[:,5]
    # rescale the coordinates
    bboxes = scale_coords(letterbox_img_curr.shape[:-1], bboxes, frame_curr.shape[:-
1]).astype(int)
    # show bbox of detections
    for bbox, score, class_id in zip(bboxes, scores, class_ids):
        color = colors[int(class_id) % len(colors)]
        cv2.rectangle(frame_curr, (bbox[0],bbox[1]), (bbox[2], bbox[3]), color, 2)
```

```
        cv2.rectangle(frame_curr, (bbox[0], bbox[1] - 20), (bbox[2], bbox[1]), color, -1)
        cv2.putText(frame_curr, class_list[class_id], (bbox[0], bbox[1] - 10),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 255, 255))
    end = time.time()
    # show FPS
    fps = (1 / (end - start))
    fps_label = "Throughput: %.2f FPS" % fps
    cv2.putText(frame_curr, fps_label, (10, 25), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
    print(fps_label+ "; Detections: " + str(len(class_ids)))
    cv2.imshow("Async API demo", frame_curr)
    # Swap the infer request, 交换当前帧推理请求和下一帧推理请求
    infer_request_curr, infer_request_next = infer_request_next, infer_request_curr
    frame_curr = frame_next
    letterbox_img_curr = letterbox_img_next
```

请读者下载：yolov5_ov2022_async_dGPU.py 和 yolov5_ov2022_sync_dGPU.py，并放入 yolov5 文件夹中，然后分别运行。

下面是上述两个程序在蝰蛇峡谷上的运行结果截图，可以清晰的看到**异步推理程序的吞吐量明显高于同步推理程序**。





1.3 结论

使用 OpenVINO Runtime 的异步推理 API，将 AI 推理程序改造为异步推理的实现方式，可以明显的提升 AI 推理程序的吞吐量。