



AI Programming Assistant

Presenter: Bob Robey
Date: Oct 21-23
AMD @ Tsukuba University

AMD 
together we advance_

What is Ollama?



Get up and running with large language models.

Run [Llama 3.3](#), [DeepSeek-R1](#), [Phi-4](#), [Mistral](#), [Gemma 3](#), and other models, locally.

Download ↓

Available for macOS, Linux,
and Windows

I asked ChatGPT:

“Ollama is a company and platform that provides an API for interacting with large language models (LLMs) and AI tools. The platform allows developers to integrate powerful AI capabilities into their applications, particularly focusing on advanced language models similar to GPT-like models.

Ollama aims to make AI tools more accessible by offering easy-to-use solutions for interacting with and deploying LLMs”

source: ollama.com

Ollama Installation

- **HPC System**

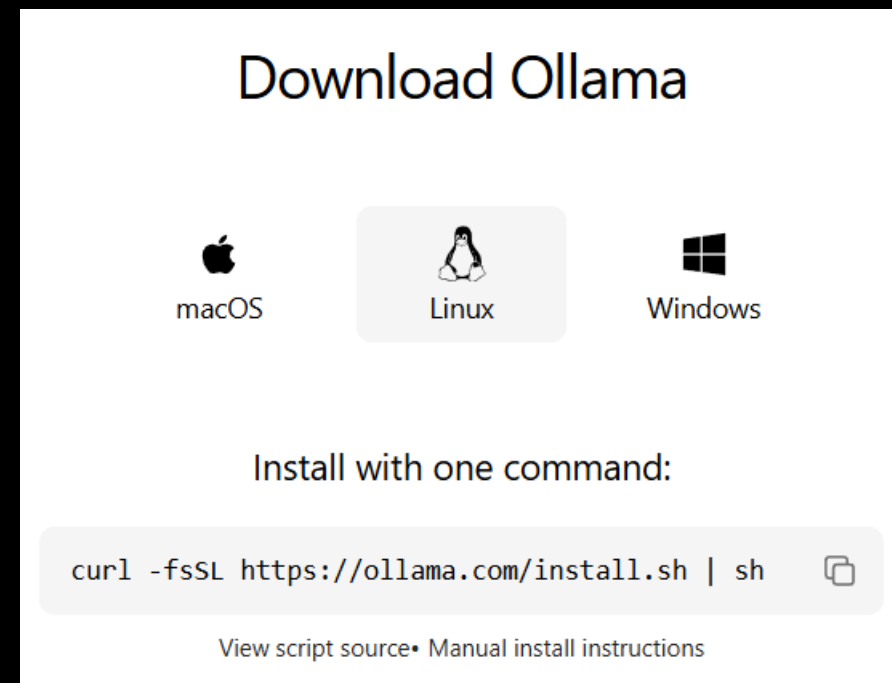
- `curl -fsSL https://ollama.com/install.sh | sh`
or maybe safer to check the commands that will be run with <https://ollama.com/install.sh>
- to run the script: `chmod +x install.sh && ./install.sh`
- manual installation also available on GitHub:
<https://github.com/ollama/ollama/blob/main/docs/linux.md>

The ollama application will be installed in `/usr/local/bin`

- **Laptop/Workstation**

- Can follow the same procedure as above
- Models that are not too big can run on a laptop without GPUs, but will be slow

source: <https://ollama.com/download/linux>



How to get the Models

- Models can be found here: <https://ollama.com/library>

models run **locally** on the cluster, workstation or laptop so no information leakage **should** be occurring

this shows the number of parameters you can use

this command shows how to run it

models themselves are open-source – no licensing, subscriptions or other fees to pay

deepseek-r1

DeepSeek's first-generation of reasoning models with comparable performance to OpenAI-o1, including six dense models distilled from DeepSeek-R1 based on Llama and Qwen.

1.5b
7b
8b
14b
32b
70b
671b

25.8M Pulls
 Updated 4 weeks ago

7b

29 Tags

ollama run deepseek-r1

Updated 7 weeks ago		0a8c26691023 · 4.7GB
model	arch qwen2 · parameters 7.62B · quantization Q4_K_M	4.7GB
params	{ "stop": ["< begin_of_sentence >", "< end_of_sentence >",	148B
template	{{- if .System }}{{ .System }}{{ end }} {{- range \$i, \$_ := .Mes...	387B
license	MIT License Copyright (c) 2023 DeepSeek Permission is hereby gra...	

Editor integration

- Vim

Vim users can use **vim-ollama**: <https://github.com/gergap/vim-ollama>

- To install:
`curl -fLo ~/.vim/autoload/plug.vim --create-dirs https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim`
- Then put this in ~/.vimrc:

```
call plug#begin()  
Plug 'gergap/vim-ollama'  
call plug#end()
```
- Open vim and do `:PlugInstall`

- Emacs

Emacs users can use **Ellama** (Emacs Large Language Model Assistant):
<https://github.com/s-kostyaev/ellama>

Prompt Engineering

- Prompt engineering means crafting and refining natural language instructions to guide AI models in generating accurate, relevant, and useful outputs.
- Several approaches:
 1. Be specific and detailed: “Write a HIP kernel that computes a reduction on an array of integers. Make sure the code works”
 2. Provide context and background Information: “I am building a shallow water model with periodic boundary conditions...”
 3. Include example code or templates (Few-Shot Learning): “Here is an example HIP kernel computing convolutions...”
 4. Specify the output format and programming language: “The code should be written in C++...”
 5. Assign a coding persona or role: “You are a C++ expert. Write a code that implements...”
 6. Break down complex tasks (Chain-of-Thought): “Step 1: recognize easily parallelizable loops. 2) Provide a parallel implementation”
 7. Use Multi-Step prompts and iterative refinement: “Parallelize the code. Next, refined the code adding error handling”
 8. Highlight constraints and best practices: “The code must show great performance, try to avoid atomic operations as much as possible.”
 9. Experiment with prompt variations: Zero/One/Few shots learning.

Example: *You are an expert [C++|C|Fortran] programmer. I want to convert a CPU code to run on an AMD GPU using OpenMP® standard 6.0. Use target directives and make sure the code works. Recognize cases when parallelism creates race conditions and take appropriate measures to avoid that.*

Coder LLM vs Reasoning LLM

Coder LLM

- For this presentation: **qwen2.5-coder:32b**
- An LLM trained on a vast amount of coding examples
- Best used for tasks where you already know the logic and just need a well-formed code snippet
- Lower verbosity is preferred to get concise code quickly.
- Helpful for repetitive and tedious tasks when syntax help is needed but logic is straightforward

Reasoning LLM

- For this presentation: **qwq:32b**
- An LLM trained to “think” using Chain-of-Thought (or similar)
- Ideal when tackling unfamiliar or complex problems where understanding the logic and process is as important as the final code
- May produce more verbose outputs that include explanations and intermediate reasoning, which can be useful for understanding the solution but might require additional parsing if you need just the code.
- Helpful when more logic needs to be used to solve a coding problem (e.g., CPU/GPU porting)

Ollama supported AMD GPUs (as of March 11 2025)

AMD Radeon

Ollama supports the following AMD GPUs:

Linux Support

Family	Cards and accelerators
AMD Radeon RX	7900 XTX 7900 XT 7900 GRE 7800 XT 7700 XT 7600 XT 7600 6950 XT 6900 XTX 6900XT 6800 XT 6800 Vega 64 Vega 56
AMD Radeon PRO	W7900 W7800 W7700 W7600 W7500 W6900X W6800X Duo W6800X W6800 V620 V420 V340 V320 Vega II Duo Vega II VII SSG
AMD Instinct	MI300X MI300A MI300 MI250X MI250 MI210 MI200 MI100 MI60 MI50

Windows Support

With ROCm v6.1, the following GPUs are supported on Windows.

Family	Cards and accelerators
AMD Radeon RX	7900 XTX 7900 XT 7900 GRE 7800 XT 7700 XT 7600 XT 7600 6950 XT 6900 XTX 6900XT 6800 XT 6800
AMD Radeon PRO	W7900 W7800 W7700 W7600 W7500 W6900X W6800X Duo W6800X W6800 V620

Source: <https://github.com/ollama/ollama/blob/main/docs/gpu.md#amd-radeon>

Start Up Ollama: gfx942 (MI300A and MI300X)

- Start ollama daemon: `ollama serve &`
 - The `&` makes it run in the background
 - Might need `export OLLAMA_HOST=127.0.0.1:11435` if getting Error: listen tcp 127.0.0.1:11434: bind: address already in use

currently issue on gfx942:

<https://github.com/ollama/ollama/issues/8735>

```
gcapodag@06a065014af1:~$ ollama serve &
[1] 3707250
gcapodag@06a065014af1:~$ 2025/03/12 03:58:04 routes.go:1215: INFO server config env="map[CUDA_VISIBLE_DEVICES: GPU_DEVICE_ORDINAL: HIP_VISIBLE_DEVICES: HSA_OV
ERRIDE_GFX_VERSION: HTTPS_PROXY: HTTP_PROXY: NO_PROXY: OLLAMA_CONTEXT_LENGTH:2048 OLLAMA_DEBUG:false OLLAMA_FLASH_ATTENTION:false OLLAMA_GPU_OVERHEAD:0 OLLAMA
_HOST:http://127.0.0.1:11435 OLLAMA_INTEL_GPU:false OLLAMA_KEEP_ALIVE:5m0s OLLAMA_KV_CACHE_TYPE: OLLAMA_LLM_LIBRARY: OLLAMA_LOAD_TIMEOUT:5m0s OLLAMA_MAX_LOADE
D_MODELS:0 OLLAMA_MAX_QUEUE:512 OLLAMA_MODELS:/home/aac/shared/teams/dcgpu_training/amd/gcapodag/.models OLLAMA_MULTIUSER_CACHE:false OLLAMA_NEW_ENGINE:false
OLLAMA_NOHISTORY:false OLLAMA_NOPRUNE:false OLLAMA_NUM_PARALLEL:0 OLLAMA_ORIGINS:[http://localhost https://localhost http://localhost:* https://localhost:* ht
tp://127.0.0.1 https://127.0.0.1 http://127.0.0.1:* https://127.0.0.1:* http://0.0.0.0 https://0.0.0.0 http://0.0.0.0:* https://0.0.0.0:* app://* file://* tau
ri://* vscode-webview://* vscode-file://*] OLLAMA_SCHED_SPREAD:false ROCR_VISIBLE_DEVICES: http_proxy: https_proxy: no_proxy:]"
time=2025-03-12T03:58:04.102Z level=INFO source=images.go:432 msg="total blobs: 0"
time=2025-03-12T03:58:04.103Z level=INFO source=images.go:439 msg="total unused blobs removed: 0"
time=2025-03-12T03:58:04.106Z level=INFO source=routes.go:1277 msg="Listening on 127.0.0.1:11435 (version 0.5.13)"
time=2025-03-12T03:58:04.107Z level=INFO source=gpu.go:217 msg="looking for compatible GPUs"
time=2025-03-12T03:58:04.167Z level=INFO source=amd_linux.go:296 msg="unsupported Radeon iGPU detected skipping" id=0 total="0 B"
time=2025-03-12T03:58:04.170Z level=INFO source=amd_linux.go:296 msg="unsupported Radeon iGPU detected skipping" id=1 total="0 B"
time=2025-03-12T03:58:04.173Z level=INFO source=amd_linux.go:296 msg="unsupported Radeon iGPU detected skipping" id=2 total="0 B"
time=2025-03-12T03:58:04.175Z level=INFO source=amd_linux.go:296 msg="unsupported Radeon iGPU detected skipping" id=3 total="0 B"
time=2025-03-12T03:58:04.175Z level=INFO source=amd_linux.go:402 msg="no compatible amdgpu devices detected"
time=2025-03-12T03:58:04.176Z level=INFO source=gpu.go:377 msg="no compatible GPUs were discovered"
time=2025-03-12T03:58:04.176Z level=INFO source=types.go:130 msg="inference compute" id=0 library=cpu variant="" compute="" driver=0.0 name="" total="502.2 Gi
B" available="443.5 GiB"
```

Start Up Ollama: gfx90a (MI200 series)

- The GPU devices are recognized for gfx90a (MI210 in this particular case)

working on gfx90a

```
gcapodag@5cbfd40d1aaa:~$ 2025/03/13 04:26:17 routes.go:1215: INFO server config env="map[CUDA_VISIBLE_DEVICES: GPU_DEVICE_ORDINAL: HIP_VISIBLE_DEVICES: HSA_OV
ERRIDE_GFX_VERSION: HTTPS_PROXY: HTTP_PROXY: NO_PROXY: OLLAMA_CONTEXT_LENGTH:2048 OLLAMA_DEBUG:false OLLAMA_FLASH_ATTENTION:false OLLAMA_GPU_OVERHEAD:0 OLLAMA
_HOST:http://127.0.0.1:11435 OLLAMA_INTEL_GPU:false OLLAMA_KEEP_ALIVE:5m0s OLLAMA_KV_CACHE_TYPE: OLLAMA_LLM_LIBRARY: OLLAMA_LOAD_TIMEOUT:5m0s OLLAMA_MAX_LOADE
D_MODELS:0 OLLAMA_MAX_QUEUE:512 OLLAMA_MODELS:/home/aac/shared/teams/dcgpu_training/amd/gcapodag/.ollama/models OLLAMA_MULTISUSER_CACHE:false OLLAMA_NEW_ENGINE
:false OLLAMA_NOHISTORY:false OLLAMA_NOPRUNE:false OLLAMA_NUM_PARALLEL:0 OLLAMA_ORIGINS:[http://localhost https://localhost http://localhost:* https://localho
st:* http://127.0.0.1 https://127.0.0.1 http://127.0.0.1:* https://127.0.0.1:* http://0.0.0.0 https://0.0.0.0 http://0.0.0.0:* https://0.0.0.0:* app://* file:
/* tauri://* vscode-webview://* vscode-file://*] OLLAMA_SCHED_SPREAD:false ROCR_VISIBLE_DEVICES: http_proxy: https_proxy: no_proxy:]"
time=2025-03-13T04:26:17.270Z level=INFO source=images.go:432 msg="total blobs: 0"
time=2025-03-13T04:26:17.271Z level=INFO source=images.go:439 msg="total unused blobs removed: 0"
time=2025-03-13T04:26:17.274Z level=INFO source=routes.go:1277 msg="Listening on 127.0.0.1:11435 (version 0.5.13)"
time=2025-03-13T04:26:17.274Z level=INFO source=gpu.go:217 msg="looking for compatible GPUs"
time=2025-03-13T04:26:17.308Z level=INFO source=amd_linux.go:386 msg="amdgpu is supported" gpu=GPU-5022e090808b5427 gpu_type=gfx90a
time=2025-03-13T04:26:17.309Z level=INFO source=amd_linux.go:386 msg="amdgpu is supported" gpu=GPU-eb8daca7c7f674a2 gpu_type=gfx90a
time=2025-03-13T04:26:17.309Z level=INFO source=amd_linux.go:386 msg="amdgpu is supported" gpu=GPU-d369dc9aab45520c gpu_type=gfx90a
time=2025-03-13T04:26:17.310Z level=INFO source=amd_linux.go:386 msg="amdgpu is supported" gpu=GPU-b598d638837782d7 gpu_type=gfx90a
time=2025-03-13T04:26:17.311Z level=INFO source=amd_linux.go:386 msg="amdgpu is supported" gpu=GPU-e32cbe3ec6f6a640 gpu_type=gfx90a
time=2025-03-13T04:26:17.312Z level=INFO source=amd_linux.go:386 msg="amdgpu is supported" gpu=GPU-ad0ed7fbc23d4c3 gpu_type=gfx90a
time=2025-03-13T04:26:17.313Z level=INFO source=amd_linux.go:386 msg="amdgpu is supported" gpu=GPU-31d0d92d2b0e6bbe gpu_type=gfx90a
time=2025-03-13T04:26:17.314Z level=INFO source=amd_linux.go:386 msg="amdgpu is supported" gpu=GPU-dfd218ad6f8e20b9 gpu_type=gfx90a
time=2025-03-13T04:26:17.314Z level=INFO source=types.go:130 msg="inference compute" id=GPU-5022e090808b5427 library=rocm variant="" compute=gfx90a driver=6.7
name=1002:740f total="64.0 GiB" available="64.0 GiB"
```

Run a Model with Ollama

- `export OLLAMA_MODELS=$HOME/.models`
`ollama run qwen2.5-coder:32b`
 - will show something like this:

Ollama run will first pull and then run, below is the output of the pull part

```
gcapodag@06a065014af1:~$ ollama run qwen2.5-coder:32b
[GIN] 2025/03/12 - 04:09:28 | 200 | 43.93µs | 127.0.0.1 | HEAD | "/"
[GIN] 2025/03/12 - 04:09:28 | 404 | 4.536189ms | 127.0.0.1 | POST | "/api/show"
pulling manifest " [GIN] 2025/03/12 - 04:09:29 | 200 | 1.148589014s | 127.0.0.1 | POST | "/api/pull"
pulling manifest
pulling ac3d1ba8aa77... 100% 19 GB
pulling 66b9ea09bd5b... 100% 68 B
pulling e94a8ecb9327... 100% 1.6 KB
pulling 832dd9e00a68... 100% 11 KB
pulling f0676bd3c336... 100% 488 B
verifying sha256 digest
writing manifest
success
```

Once ready, you will see this prompt:

```
>>> Send a message (/? for help)
```

To exit, type /bye

Hipify a code with qwen2.5-coder:32b

- Hipify this code:
HPCTrainingExamples/HIP/vector_addition_examples/vector_addition_hipify/vector_addition.cu
- Prompt:
You are an experienced GPU programmer with knowledge of AMD GPUs and HIP, hipify this code: <code>
- Answer:

To convert the given CUDA code to use HIP (Heterogeneous-computing Interface for Portability) for AMD GPUs, you need to replace CUDA-specific API calls and syntax with their HIP equivalents. Below is the hipified version of your code:

...
output code
...

Key Changes:

1. **Kernel Launch Syntax**: The `<<<...>>>` syntax is replaced with `hipLaunchKernelGGL`.
2. **API Function Names**: CUDA API functions are prefixed with `cuda` and need to be replaced with the corresponding HIP functions prefixed with `hip`, such as `cudaMalloc` becomes `hipMalloc`.
3. **Error Checking**: The `error checking` macro remains mostly the same but uses `hipGetErrorString` instead of `cudaGetErrorString`.

This code should now be compatible with AMD GPUs using the HIP framework. Make sure to compile it with a HIP-capable compiler, such as `hcc` or `clang` with HIP support **enabled**.

Hipify a code with qwen2.5-coder:32b

- Compare with:
HPCTrainingExamples/HIP/vector_addition_examples/vector_addition_error_check/vector_addition.cpp

Forgot to
add the
hip_runtime
include,
but so did
hipify-perl

qwen2.5-coder

Using an old
syntax but
still working

```
#include <stdio.h>
#include <math.h>
/* Macro for checking GPU API return values */
#define gpuCheck(call)
do{
    hipError_t gpuErr = call;
    if(hipSuccess != gpuErr){
        printf("GPU API Error - %s:%d: '%s'\n", __FILE__, __LINE__, hipGetErrorString(gpuErr));
        exit(1);
    }
}while(0)

/* -----
Vector addition kernel
----- */
global void vector_addition(double *A, double *B, double *C, int n)
+-- 40 lines: {
    thr_per_blk: number of threads per thread block
    blk_in_grid: number of thread blocks in grid */
    int thr_per_blk = 256;
    int blk_in_grid = ceil( float(N) / thr_per_blk );

    /* Launch vector addition kernel */
    hipLaunchKernelGGL(vector_addition, dim3(blk_in_grid), dim3(thr_per_blk),
    /* Check for kernel launch errors */
    gpuCheck( hipGetLastError() );

    /* Check for kernel execution errors */
    gpuCheck ( hipDeviceSynchronize() );
+-- 31 lines: Copy data from device array to host array (only need result, d
printf("Blocks in Grid : %d\n", blk_in_grid);
printf("Threads per Block: %d\n", thr_per_blk);
printf("=====\n\n");

    return 0;
}
```

```
#include <stdio.h>
#include <math.h>
#include "hip/hip_runtime.h"
/* Macro for checking GPU API return values */
#define gpuCheck(call)
do{
    hipError_t gpuErr = call;
    if(hipSuccess != gpuErr){
        printf("GPU API Error - %s:%d: '%s'\n", __FILE__, __LINE__, hipGetErrorString(gpuErr));
        exit(1);
    }
}while(0)

/* -----
Vector addition kernel
----- */
global void vector_addition(double *A, double *B, double *C, int n)
+-- 40 lines: {
    thr_per_blk: number of threads per thread block
    blk_in_grid: number of thread blocks in grid */
    int thr_per_blk = 256;
    int blk_in_grid = ceil( float(N) / thr_per_blk );

    /* Launch vector addition kernel */
    vector_addition<<<blk_in_grid, thr_per_blk>>>>(d_A, d_B, d_C, N);
    /* Check for kernel launch errors */
    gpuCheck( hipGetLastError() );

    /* Check for kernel execution errors */
    gpuCheck ( hipDeviceSynchronize() );
+-- 31 lines: Copy data from device array to host array (only need result, d
printf("Blocks in Grid : %d\n", blk_in_grid);
printf("Threads per Block: %d\n", thr_per_blk);
printf("=====\n\n");

    return 0;
}
```

original

Hipify a code with qwq:32b

- Hipify this code:
HPCTrainingExamples/HIP/vector_addition_examples/vector_addition_hipify/vector_addition.cu
- Prompt:
You are an experienced GPU programmer with knowledge of AMD GPUs and HIP, hipify this code: <code>
- Answer:

So, first step: Include `<hip/hip_runtime.h>` instead of the implicit `cuda_runtime.h`. The user's original code didn't include that explicitly, but CUDA code typically includes `#include <cuda_runtime.h>`. So adding:

```
#include <hip/hip_runtime.h>
```

At the top, replacing any CUDA-specific headers.



it knows about the `hip_runtime`

but it **does not** produce code

Now, looking at all instances of 'cuda' in the code:

- The macro uses `cudaError_t` → change to `hipError_t`
 - `cudaMalloc` → `hipMalloc`
 - `cudaMemcpyHostToDevice` → `hipMemcpyHostToDevice` (the enum is the same but under HIP)
- Wait: `cudaMemcpyHostToDevice` becomes `hipMemcpyHostToDevice`. Similarly for `DeviceToHost`.

So all `cudaMemcpy` calls should have their direction enums prefixed with 'hip' instead of 'cuda'.

The line:

```
gpuCheck( cudaMemcpy(d_A, h_A, bytes, cudaMemcpyHostToDevice) );
```

Becomes:

```
gpuCheck( hipMemcpy(d_A, h_A, bytes, hipMemcpyHostToDevice) );
```

Same for the other `cudaMemcpy` calls.

Write a portable CMakeLists.txt with qwen2.5-coder:32b

- Write a portable CMakeLists.tx for this code: HPCTrainingExamples/HIP/vectorAdd/vectoradd.hip
- Prompt:
 - You are an experienced CMAKE programmer with knowledge of HIP and CUDA, write a CMakeLists.txt that is portable between HIP and CUDA when the user specifies CMAKE_GPU_RUNTIME as input variable for CMake. Note that HIP is a portable language so you do not need to switch source file for CUDA or HIP, the same source file called “vectoradd.hip” can be use for either choice of CMAKE_GPU_RUNTIME, this is the code: <code>
- Answer:

To create a `CMakeLists.txt` that can handle both HIP and CUDA based on the `CMAKE_GPU_RUNTIME` variable, you need to set up your CMake project to conditionally include the appropriate compiler flags and link against the correct libraries. The same source file `vectoradd.hip` will be used for both HIP and CUDA.

Here's a CMakeLists.txt that achieves this:

Usage:

To build with HIP, you would run CMake like this:

```
```sh
cmake -DCMAKE_GPU_RUNTIME=HIP ..
```
```

To build with CUDA, you would run:

```
```sh
cmake -DCMAKE_GPU_RUNTIME=CUDA ..
```
```

Compare with:

HPCTrainingExamples/HIP/vectorAdd/CMakeLists.txt

Write a portable CMakeLists.txt with qwen2.5-coder:32b

How was the code for HIP?

- Just needed to modify one line: `project(VectorAdd LANGUAGES CXX CUDA ${CMAKE_GPU_RUNTIME})`
but only so we could run it on a system where CUDA is not installed

```
gcapodag@06a065014af1:~/HPCTrainingExamples/HIP/vectorAdd/build$ cmake -DCMAKE_GPU_RUNTIME=HIP ..
-- The HIP compiler identification is Clang 18.0.0
-- Detecting HIP compiler ABI info
-- Detecting HIP compiler ABI info - done
-- Check for working HIP compiler: /opt/rocm-6.3.3/lib/llvm/bin/clang++ - skipped
-- Detecting HIP compile features
-- Detecting HIP compile features - done
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Configuring done (2.6s)
-- Generating done (0.1s)
-- Build files have been written to: /home/aac/shared/teams/dcgpu_training/amd/gcapodag/HPCTrainingExamples/HIP/vectorAdd/build
gcapodag@06a065014af1:~/HPCTrainingExamples/HIP/vectorAdd/build$ make
[ 50%] Building HIP object CMakeFiles/vector_add.dir/vectoradd.hip.o
[100%] Linking HIP executable vector_add
[100%] Built target vector_add
gcapodag@06a065014af1:~/HPCTrainingExamples/HIP/vectorAdd/build$ ./vector_add
System minor 4
System major 9
agent prop name AMD Instinct MI300A
hip Device prop succeeded
PASSED!
```

```
gcapodag@06a065014af1:~/HPCTrainingExamples/HIP/vectorAdd/build$ ./vector_add
System minor 4
System major 9
agent prop name AMD Instinct MI300A
hip Device prop succeeded
PASSED!
```

Write a portable CMakeLists.txt with qwen2.5-coder:32b

How was the code for CUDA?

Needed to add:

- `find_package(CUDAToolkit REQUIRED)`
- ```
if (${CMAKE_GPU_RUNTIME} MATCHES "CUDA")
 set(CMAKE_CUDA_FLAGS "${CMAKE_CUDA_FLAGS} -I/$ENV{HIP_PATH}/include")
endif()
```
- `set_source_files_properties(${SOURCE_FILES} PROPERTIES LANGUAGE ${CMAKE_GPU_RUNTIME})`

# Write a portable CMakeLists.txt with qwen2.5-coder:32b

How was the code for CUDA?

Tested on Gilgamesh (University of Oregon system with NVIDIA H100 and AMD MI210)

```
[gcapodag@gilgamesh build]$ cmake -DCMAKE_GPU_RUNTIME=CUDA ..
-- The CXX compiler identification is GNU 8.5.0
-- The CUDA compiler identification is NVIDIA 12.8.93
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Detecting CUDA compiler ABI info
-- Detecting CUDA compiler ABI info - done
-- Check for working CUDA compiler: /packages/cuda/12.8.1/bin/nvcc - skipped
-- Detecting CUDA compile features
-- Detecting CUDA compile features - done
-- Found CUDAToolkit: /packages/cuda/12.8.1/include (found version "12.8.93")
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Failed
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
-- Configuring done (2.5s)
-- Generating done (0.0s)
-- Build files have been written to: /home/users/gcapodag/repos/HPCTrainingExamples/HIP/vectorAdd/build
```

Shows that we are  
running on Nvidia

```
[gcapodag@gilgamesh build]$./vector_add
System minor 0
System major 9
agent prop name NVIDIA H100 PCIe
hip Device prop succeeded
PASSED!
```

# Write OpenMP® target directives with qwen2.5-coder:32b

- Start from this code:  
Pragma\_Examples/OpenMP/Fortran/1\_saxpy/0\_saxpy\_serial\_portyourself/saxpy.F90
- Prompt:  
You are an expert Fortran and GPU programmer. You want to convert a CPU code to run on an AMD GPU using OpenMP standard 6.0. Use target directives and make sure the code works. Recognize cases when parallelism creates race conditions and take appropriate measures to avoid that. The CPU code is: <code>
- Answer:

To convert your CPU code to run on an AMD GPU using OpenMP 6.0 and its target directives, you need to make several modifications. The key changes involve adding `!$omp target` and related directives to offload the work to the GPU. Additionally, you should ensure that any parallelism does not introduce race conditions.

Here's how you can modify your code:

1. **Add Target Directives**: Use `!$omp target` to specify which parts of the code should run on the GPU.
2. **Data Mapping**: Ensure data is mapped correctly between the host (CPU) and device (GPU).
3. **Avoid Race Conditions**: In this specific case, there are no race conditions since each element of `y` is updated independently.

Here's the modified code:

# Write OpenMP® target directives with qwen2.5-coder:32b

How was the code?

- It forgot to declare the loop index `i: integer :: i`
- Compiler warning: `simd` information on composite construct discarded (that is just due to `amdflang-new`)

This is the key part of the code

```
!$omp target map(to: a,x) map(tofrom: y)
!$omp teams distribute parallel do simd
do i=1,n
 y(i) = a * x(i) + y(i)
end do
!$omp end teams distribute parallel do simd
!$omp end target
```



I modified it like this and got the same exact output

```
!$omp target teams distribute parallel do simd map(to: a,x) map(tofrom: y)
do i=1,n
 y(i) = a * x(i) + y(i)
end do
!$omp end target teams distribute parallel do simd
```

```
gcapodag@06a065014af1:~/HPCTrainingExamples/Pragma_Examples/OpenMP/Fortran/1_saxpy/0_saxpy_serial_portyourself$./ollama
DEVID: 0 SGN:2 ConstWGSize:256 args: 8 teamsXthrs:(1368X 256) reqd:(0X 0) lds_usage:0B sgpr_count:20 vgpr_count:10 sgpr_spill_count:0 vgpr_spill_count
:0 tripcount:10000000 rpc:0 md:0 md_LB:-1 md_UB:-1 Max Occupancy: 8 Achieved Occupancy: 75% n:__omp_offloading_32_5280b_QMsaxpymodPinititalize_l47
DEVID: 0 SGN:2 ConstWGSize:256 args: 9 teamsXthrs:(1368X 256) reqd:(0X 0) lds_usage:0B sgpr_count:22 vgpr_count:14 sgpr_spill_count:0 vgpr_spill_count
:0 tripcount:10000000 rpc:0 md:0 md_LB:-1 md_UB:-1 Max Occupancy: 8 Achieved Occupancy: 75% n:__omp_offloading_32_5280b_QMsaxpymodPsaxpy_l23
Time of kernel: 0.000267
plausibility check:
y(1) 4.
y(n-1) 4.
```

# Write OpenMP® target directives with qwen2.5-coder:32b

Next level:

- Prompt:

If you are on an MI300A with HSA\_XNACK=1, what could be simplified in the program?

- Answer:

When working on an AMD MI300A GPU with `HSA_XNACK=1`, you can simplify some aspects of your OpenMP target directives. The `HSA_XNACK=1` setting allows for the handling of non-accessible memory (XNACK) in Heterogeneous System Architecture (HSA), which means that you don't need to explicitly map data between host and device using the `map` clause as frequently. This can simplify your code.

Here's how you can adjust your program to take advantage of `HSA_XNACK=1`:

1. **\*\*Remove Explicit Data Mapping\*\***: Since `HSA_XNACK=1` handles memory access automatically, you can omit the explicit `map` clauses.
2. **\*\*Simplify Target Directives\*\***: Focus on offloading the loops and parallelism without worrying about data mapping.

Here's the simplified version of your code:

# Write OpenMP® target directives with qwen2.5-coder:32b

How was the code this time?

- Still forgot to declare the loop index `i: integer :: i`
- Same compiler warning: `simd` information on composite construct discarded (that is just due to `amdflang-new`)

## Before

```
!$omp target map(to: a,x) map(tofrom: y)
!$omp teams distribute parallel do simd
do i=1,n
 y(i) = a * x(i) + y(i)
end do
!$omp end teams distribute parallel do simd
!$omp end target
```

## After

```
!$omp target
!$omp teams distribute parallel do simd
do i=1,n
 y(i) = a * x(i) + y(i)
end do
!$omp end teams distribute parallel do simd
!$omp end target
```

## Same output as before but faster

```
gcapodag@06a065014af1:~/HPCTrainingExamples/Pragma_Examples/OpenMP/Fortran/1_saxpy/3_saxpy_parallel dosimd$./ollama
DEVID: 0 SGN:2 ConstWGSsize:256 args: 6 teamsXthrds:(1368X 256) reqd:(0X 0) lds_usage:0B sgpr_count:20 vgpr_count:10 sgpr_spill_count:0 vgpr_spill_count
:0 tripcount:10000000 rpc:0 md:0 md_LB:-1 md_UB:-1 Max Occupancy: 8 Achieved Occupancy: 75% n:__omp_offloading_32_1a359_QMsaxpymodPinitialize_l46
DEVID: 0 SGN:2 ConstWGSsize:256 args: 6 teamsXthrds:(1368X 256) reqd:(0X 0) lds_usage:0B sgpr_count:22 vgpr_count:13 sgpr_spill_count:0 vgpr_spill_count
:0 tripcount:10000000 rpc:0 md:0 md_LB:-1 md_UB:-1 Max Occupancy: 8 Achieved Occupancy: 75% n:__omp_offloading_32_1a359_QMsaxpymodPsaxpy_l23
Time of kernel: 0.000127
y(1) 4.
y(n-1) 4.
```

# Convert OpenACC to OpenMP® code with qwen2.5-coder:32b

- Start from this code: Pragma\_Examples/OpenACC/Fortran/freduce/freduce.F

```
do i=1, 10
 sum2=0.0
 sum1=1000.0
 !$ACC PARALLEL LOOP REDUCTION(+:sum2) COPYIN(array(1:10))
 do j=1, 10
 sum2=sum2+array(j)
 end do

 buffer(i)=sum1+sum2
end do
```

relevant part

Prompt:

You are an experienced Fortran programmer with knowledge of OpenACC and OpenMP, convert the following code that performs a reduction with OpenACC into a code that uses OpenMP to perform the same operation but on a GPU. This is the code: <code>

- Answer:

Converting a code from using OpenACC to OpenMP for GPU execution involves several changes. However, it's **important to note** that **\*\*OpenMP does not natively support GPU offloading\*\*** as of the latest standards (as of 2023). OpenMP primarily targets multi-threaded CPU execution with some support for accelerators through directives like ``target``, but this requires specific compiler support and configurations.

If you want to perform a reduction operation on a GPU using OpenMP, you would typically use an extension or a combination of OpenMP and another tool that supports GPU offloading. One common approach is to use **\*\*NVIDIA's OpenMP target extensions\*\*** with the NVIDIA HPC SDK (formerly known as PGI compilers).

Below is a step-by-step guide to convert your Fortran code from using OpenACC to using OpenMP with GPU offloading, assuming you have access to a compiler that supports these features.

# Convert OpenACC to OpenMP<sup>®</sup> code with qwen2.5-coder:32b

How was the code?

- Getting compiler errors but it somewhat understood the assignment

qwen2.5-coder:32b

```
!$omp target map(to: array) map(from: buffer)
do i=1, 10
 sum2 = 0.0
 sum1 = 1000.0
 !$omp teams distribute parallel do reduction(+:sum2)
 do j=1, 10
 sum2 = sum2 + array(j)
 end do

 buffer(i) = sum1 + sum2
end do
!$omp end target
```

revision

```
do i=1, 10
 sum2 = 0.0
 sum1 = 1000.0
 !$omp target teams distribute parallel do reduction(+:sum2) map(to: array) map(from: buffer)
 do j=1, 10
 sum2 = sum2 + array(j)
 end do
 !$omp end target teams distribute parallel do

 buffer(i) = sum1 + sum2
end do
```

compile error due to `!$omp teams` not being after `!$omp target`

```
error: loc("/home/aac/shared/teams/dcgpu_training/amd/gcapodag/HPCTrainingExamples/Pragma_Examples/OpenACC/Fortran/freduce/ollama.F90":20:15): expected to be
nested inside of omp.target or not nested in any OpenMP dialect operations
error: verification of lowering to FIR failed
```

# Convert OpenACC to OpenMP<sup>®</sup> code with qwen2.5-coder:32b

- Start from this code: `HPCTrainingExamples/Pragma_Examples/OpenACC/Fortran/freduce/freduce.F`
- Prompt:  
You are an experienced Fortran programmer with knowledge of OpenACC and OpenMP, convert the following code that performs a reduction with OpenACC into a code that uses OpenMP to perform the same operation but on a GPU. **Note that we only want the reduction to happen on GPU.** This is the code: `<code>`
- Answer:

To convert the given Fortran code from using OpenACC to using OpenMP with GPU offloading for the reduction operation, you can use the ``target teams distribute parallel for`` construct provided by OpenMP 4.5 and later. Here's how you can modify the code:

```
!$omp target data map(to: array) map(from: buffer)
do i=1, 10
 sum2=0.0
 sum1=1000.0

 ! Offload the reduction to the GPU
 !$omp target teams distribute parallel do reduction(+:sum2)
 do j=1, 10
 sum2=sum2+array(j)
 end do

 buffer(i)=sum1+sum2
end do
!$omp end target data
```

the code compiles and runs  
out of the box with  
`amdfang-new` and **`HSA_XNACK=1`**

note the use of `!$omp target data map`  
instead of `!$omp target map`

# What about Reproducibility?

I gave the exact same prompt and code as in the previous slide to qwen2.5-coder:32b, this time running on MI210 instead of MI300A

Answer:

To convert the given Fortran code that uses OpenACC to a version using OpenMP for GPU execution, you need to make use of the ``target`` directive provided by OpenMP. The ``target`` directive allows offloading loops and other constructs to the GPU. However, **note** that OpenMP does not directly support reduction on the GPU in the same way as OpenACC. You will need to manually handle the reduction using shared memory within a GPU kernel.

## Previous result

```
!$omp target data map(to: array) map(from: buffer)
do i=1, 10
 sum2=0.0
 sum1=1000.0

 ! Offload the reduction to the GPU
 !$omp target teams distribute parallel do reduction(+:sum2)
 do j=1, 10
 sum2=sum2+array(j)
 end do

 buffer(i)=sum1+sum2
end do
!$omp end target data
```

## Current result

```
!$omp target data map(to:array) map(fromto:buffer)
do i=1, 10
 sum2 = 0.0
 sum1 = 1000.0

 ! Offload the loop to the GPU with manual reduction
 !$omp target teams distribute parallel do private(j) reduction(+:sum2)
 do j=1, 10
 sum2 = sum2 + array(j)
 end do

 buffer(i) = sum1 + sum2
end do
!$omp end target data
```

compiler error

not needed

# Convert complex C++ code with MPI and OpenMP® to Fortran

- Start from this code:  
HPCTrainingExamples/MPI-examples/GhostExchange/GhostExchange\_ArrayAssign/Ver1/GhostExchange.cc
- Prompt:  
You are an experienced Fortran and C++ programmer with knowledge of OpenMP to run on GPU, MPI and GPU programming. The following code is computing a simple averaging kernel over a 2D domain discretized with a Cartesian structured mesh which is partitioned among MPI ranks. The code executes in parallel with MPI and OpenMP and is currently implemented in C++. Note that the code is using an allocator for 2D arrays called malloc2D.h. Consider that in Fortran, handling 2D arrays is a lot simpler than in C++ and therefore do not assume you need any external allocator for 2D arrays. Note additionally that the following code is written for AMD GPUs with HSA\_XNACK=1 and is using unified share memory. Using this information, **write a variant of this code** that does the same job, but is written in Fortran instead of C++, here is the code: <code>
- Answer:

The provided C code simulates a distributed Cartesian grid using MPI, with each process handling a subgrid of a larger global grid. The code includes functionalities to initialize the grid, perform halo exchanges between neighboring processes, and print the entire grid in a structured format. Below is an analysis and explanation of key parts of the code:

### Key Components  
### Example Usage

### Key MPI Functions Used  
### Potential Improvements

By understanding these components and their interactions, you can effectively use and extend this code for distributed computing tasks involving Cartesian grids. [GIN] 2025/03/12 - 21:50:06 | 200 | 3m43s | 127.0.0.1 | POST "/api/chat"

# A note on the legal implications of using AI-generated code

## 1. Copyright and Ownership

- No Human, No Copyright: In the U.S. and EU, AI-generated code without human authorship is not protected by copyright – it defaults to public domain
- Partial Protection: If a human modifies AI output significantly, they may claim copyright over those modifications but not the raw AI output

## 2. Attribution & Licensing Risks

- Open-Source Compliance: AI may generate GPL, MIT, or Apache-licensed code, triggering obligations (e.g., attribution, license adherence)

## 3. Infringement & Legal Risks

- Direct Copying: AI may reproduce verbatim or near-verbatim segments from training data, risking copyright infringement
- Derivative Works: Modifying or translating protected code via AI may be an infringement, if the AI output is too similar to the original



**Takeaway:** AI-generated code **may not be copyrightable** but **can still infringe copyrights**. Treat AI as an assistant, **not an autonomous coder**

# Summary

- LLMs can assist programmers with some of their computing task
- Coder models can produce code as advised by the prompt, limited number of information shared, focusing on key aspects
- Reasoning models can provide insights on a topic and serve as a “source of information”, but tend to be rather chatty
- Rather simple tasks can be processed effectively, more complex task not there yet
- The prompt provided can make a big difference in the outcome obtained
- LLMs can assist programmers with conversion tools that may not be widely available (such as OpenACC to OpenMP<sup>®</sup> conversion)
- Outcome has to be viewed in probabilistic sense, so reproducibility is not really to be expected

# Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS.' AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Third-party content is licensed to you directly by the third party that owns the content and is not licensed to you by AMD. ALL LINKED THIRD-PARTY CONTENT IS PROVIDED "AS IS" WITHOUT A WARRANTY OF ANY KIND. USE OF SUCH THIRD-PARTY CONTENT IS DONE AT YOUR SOLE DISCRETION AND UNDER NO CIRCUMSTANCES WILL AMD BE LIABLE TO YOU FOR ANY THIRD-PARTY CONTENT. YOU ASSUME ALL RISK AND ARE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY ARISE FROM YOUR USE OF THIRD-PARTY CONTENT.

© 2025 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, AMD CDNA, AMD ROCm, AMD Instinct, and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.

qwq and qwen2.5-coder are open-source models from Alibaba Cloud

The OpenMP® name and the OpenMP® logo are registered trademarks of the OpenMP Architecture Review Board

