OAK RIDGE
National Laboratory

ChatHPC: Building the foundation of a new era of AI-assisted HPC

Keita Teranishi

Group Leader, teranishik@ornl.gov

U.S. DEPARTMENT OF
ENERGY

ORNL IS MANAGED BY UT-BATTELLE LLC
FOR THE US DEPARTMENT OF ENERGY

Acknowledgement

- Pedro Valero Lara
- William Godoy
- Aaron Young
- Prasanna Balaprakash
- Jeffery Vetter

- US DOE ASCR AI for Science, Next Generation Scientific Software Technology, ScIDAC RAPIDS-2

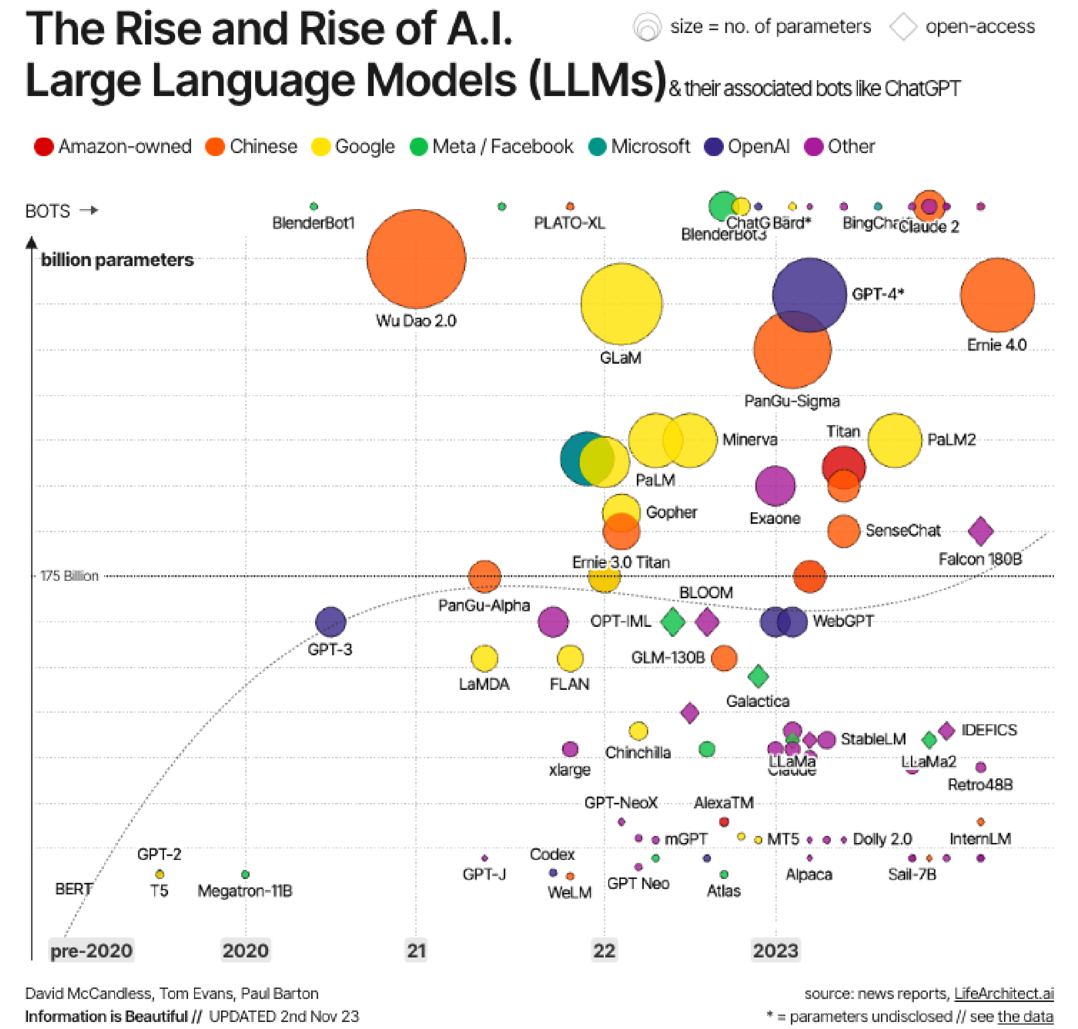
Background

Large Language Model has been major vehicle of the AI technology

- **GPT-4:** Generative Pre-trained Transformer 4 is a LLM released by OpenAI in 2023. More than 1T parameters

Already applied to coding and software engineering

- **CodeLlama:** LLM released by Meta in 2023 for coding. 70B parameters
- **GitHub Copilot:** Microsoft AI pair programmer
- **OpenAI Codex:** descendant of GPT-3 model behind GitHub Copilot
- **ChatGPT**: chatbot by OpenAI launched in November 2022. Powered by GPT-3.5/4



The Rise and Rise of A.I.
Large Language Models (LLMs) & their associated bots like ChatGPT

David McCandless, Tom Evans, Paul Barton
Information is Beautiful // UPDATED 2nd Nov 23

source: news reports, LifeArchitect.ai
* = parameters undisclosed // see the data

Motivation: Improve Programming Productivity with LLMs

- ***Define the fundamental practices and criteria to interact with LLMs for HPC targets to elevate the trustworthiness and performance levels of the AI-generated HPC codes***
- X = time required by a developer to implement a piece of code
- Y = time required by a developer to implement the same piece of code with the help of LLMs

Our target is to make Y<<X

--- Not generate perfect codes (for the moment ...)---

Techniques

- Prompt Engineering
  - While a prompt is natural language text describing the task that an AI should perform, prompt engineering is the process of structuring an instruction that can be interpreted and understood by a generative AI model.
- Multimodal Prompting
  - Multimodal prompts are a type of prompts that combine multiple input type formats, in our case: text + source code.
  - Automatic parallelization and automatic translation of HPC/scientific codes.
- Fine-tuning (Training)
  - Transfer learning in which the parameters of a pre-trained model are trained on new data.

Automatic Parallelization with LLMs

Automatic Parallelization

- OpenAI’s ChatGPT interactive chatbot v3.5: https://chat.openai.com/
- Results are correct or not correct

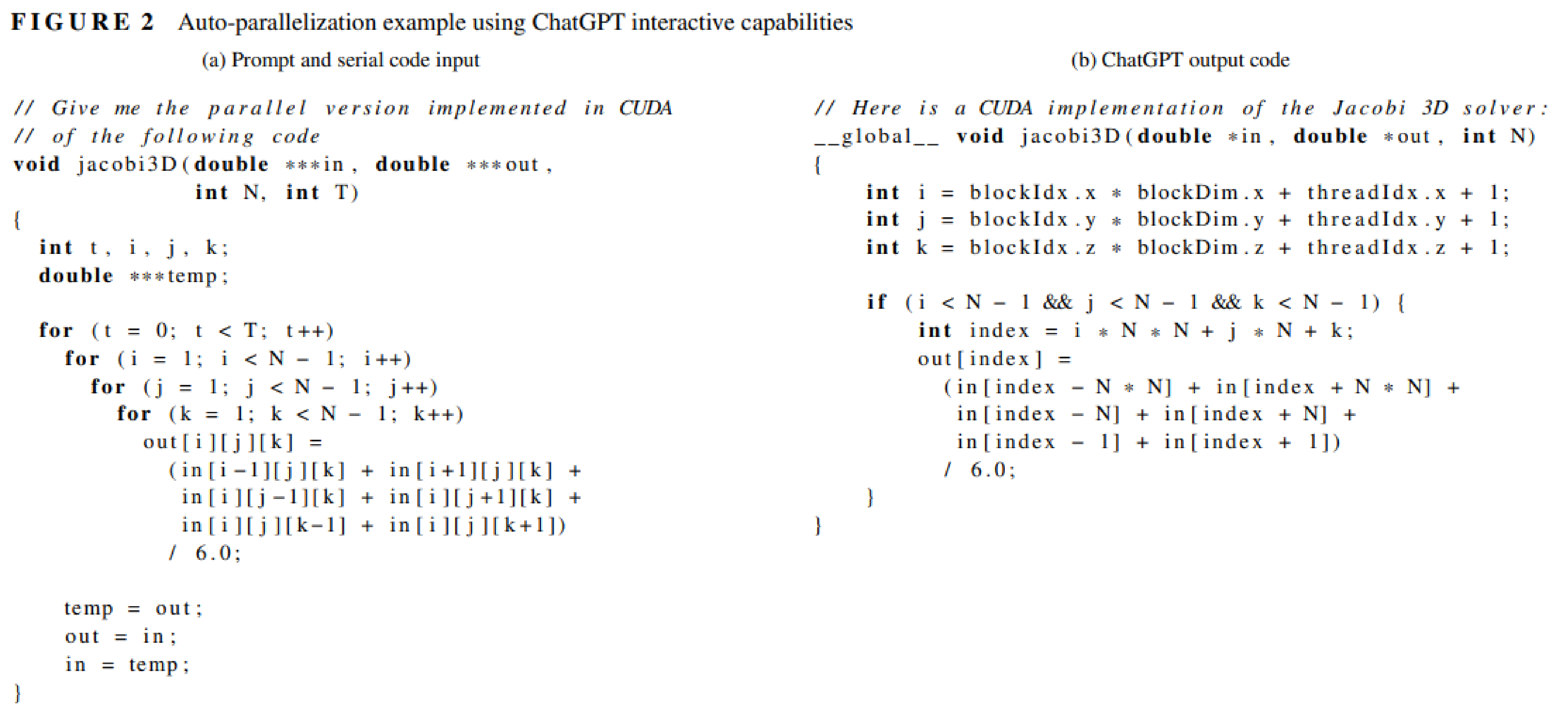


FIGURE 2 Auto-parallelization example using ChatGPT interactive capabilities

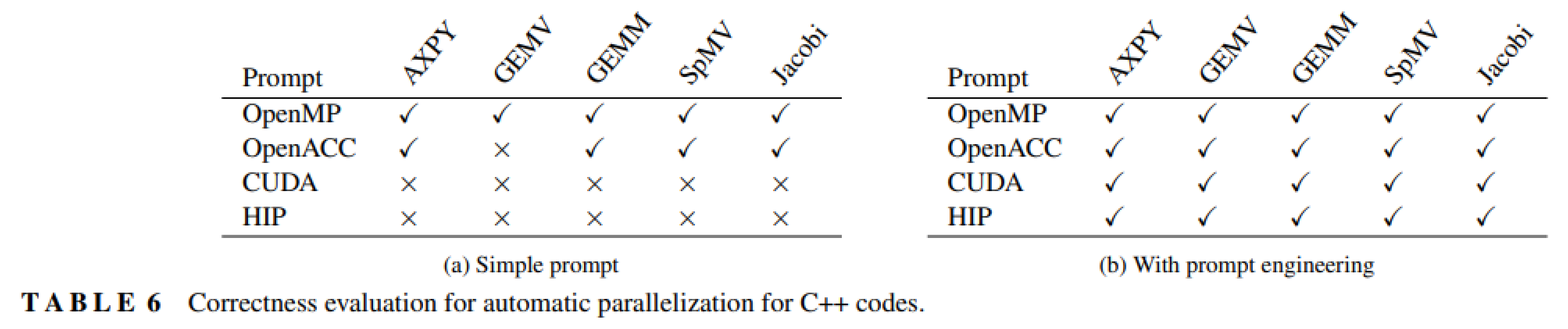
(a) Prompt and serial code input

```
// Give me the parallel version implemented in CUDA
// of the following code
void jacobi3D(double ***in, double ***out,
              int N, int T)
{
  int t, i, j, k;
  double ***temp;

  for (t = 0; t < T; t++)
    for (i = 1; i < N - 1; i++)
      for (j = 1; j < N - 1; j++)
        for (k = 1; k < N - 1; k++)
          out[i][j][k] =
            (in[i-1][j][k] + in[i+1][j][k] +
             in[i][j-1][k] + in[i][j+1][k] +
             in[i][j][k-1] + in[i][j][k+1])
            / 6.0;

    temp = out;
    out = in;
    in = temp;
}
```

(b) ChatGPT output code

```
// Here is a CUDA implementation of the Jacobi 3D solver:
__global__ void jacobi3D(double *in, double *out, int N)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x + 1;
    int j = blockIdx.y * blockDim.y + threadIdx.y + 1;
    int k = blockIdx.z * blockDim.z + threadIdx.z + 1;

    if (i < N - 1 && j < N - 1 && k < N - 1) {
        int index = i * N * N + j * N + k;
        out[index] =
          (in[index - N * N] + in[index + N * N] +
           in[index - N] + in[index + N] +
           in[index - 1] + in[index + 1])
          / 6.0;
    }
}
```

Automatic Parallelization: C++



Prompt	AXPY	GEMV	GEMM	SpMV	Jacobi
OpenMP	✓	✓	✓	✓	✓
OpenACC	✓	×	✓	✓	✓
CUDA	×	×	×	×	×
HIP	×	×	×	×	×

(a) Simple prompt

Prompt	AXPY	GEMV	GEMM	SpMV	Jacobi
OpenMP	✓	✓	✓	✓	✓
OpenACC	✓	✓	✓	✓	✓
CUDA	✓	✓	✓	✓	✓
HIP	✓	✓	✓	✓	✓

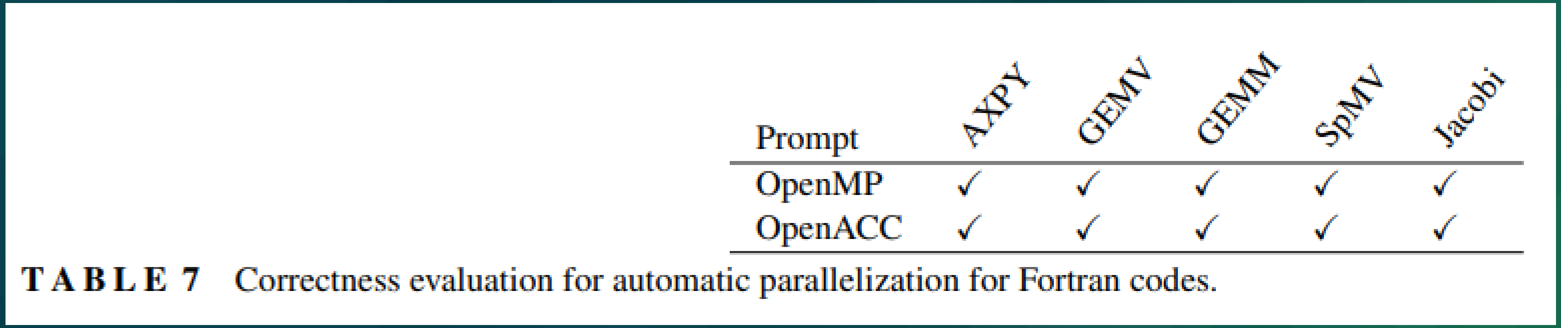
(b) With prompt engineering

TABLE 6 Correctness evaluation for automatic parallelization for C++ codes.

- For OpenACC GEMV, we try to correct the defective use of the collapse clause: "***Give me the parallel version implemented in OpenACC without using collapse of the following code . . .*** ".

- For CUDA and HIP programming models, we add more relevant information to achieve the desired correct result: "***Give me the parallel version implemented in HIP including memory allocations and copies for GPU memory of the following code, and using the corresponding CUDA/HIP API for block and thread identifications . . .*** "

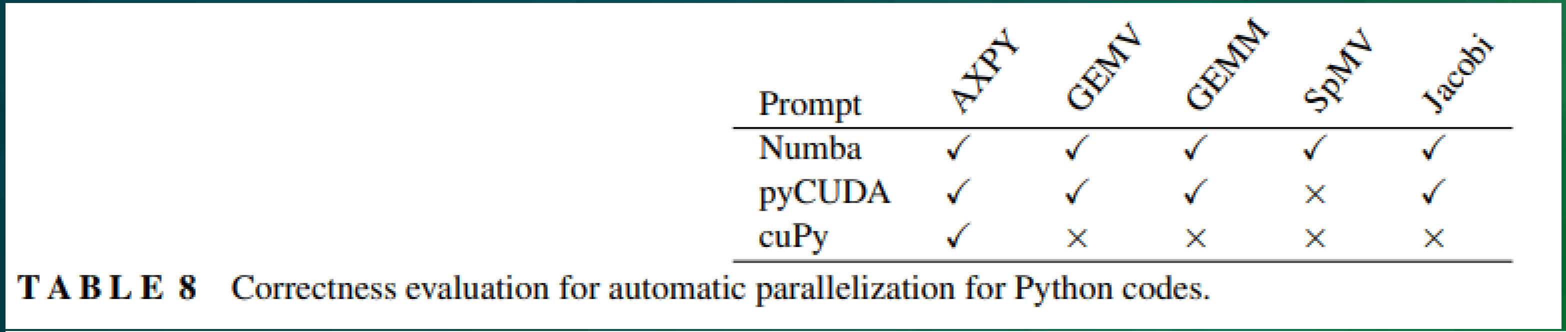
Automatic Parallelization: Fortran



Prompt	AXPY	GEMV	GEMM	SpMV	Jacobi
OpenMP	✓	✓	✓	✓	✓
OpenACC	✓	✓	✓	✓	✓

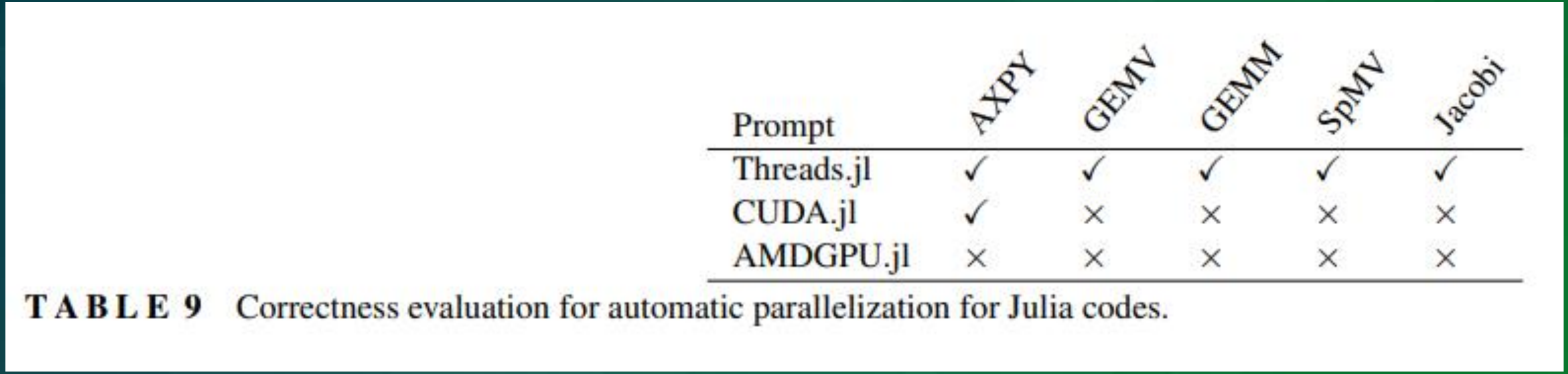
TABLE 7 Correctness evaluation for automatic parallelization for Fortran codes.

Automatic Parallelization: Python and Julia



Prompt	AXPY	GEMV	GEMM	SpMV	Jacobi
Numba	✓	✓	✓	✓	✓
pyCUDA	✓	✓	✓	×	✓
cuPy	✓	×	×	×	×

TABLE 8 Correctness evaluation for automatic parallelization for Python codes.



Prompt	AXPY	GEMV	GEMM	SpMV	Jacobi
Threads.jl	✓	✓	✓	✓	✓
CUDA.jl	✓	×	×	×	×
AMDGPU.jl	×	×	×	×	×

TABLE 9 Correctness evaluation for automatic parallelization for Julia codes.

Conclusions

Providing sequential codes to be parallelized can provide better results than code generation

- C++
    - We can get 100% efficiency (100% results are correct) by prompt engineering
- Fortran
    - 100% efficiency. No prompt engineering needed
    - Only high-level (OpenMP and OpenACC) models
- Python and Julia
    - Poor results
    - Lack of training in these languages
    - Julia is not as mature as other languages
    - Python is not widely used for HPC

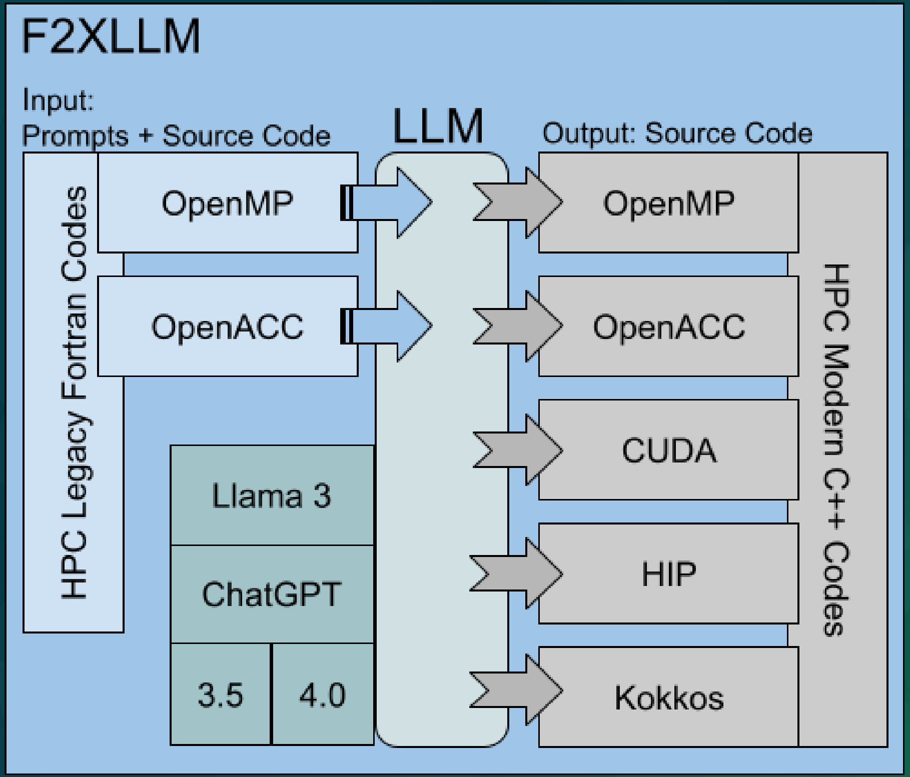
Code Translation with LLMs

F2XLLM: Modernizing HPC Fortran Codes with LLMs

Motivation

- Modernize legacy HPC Fortran codes
  - Translating HPC Fortran codes to HPC C/C++ codes using LLMs (F2XLLM)
  - HPC capability in Fortran is not at the level of modern programming languages such as C/C++
  - GPU acceleration on modern hardware or modern performance portability capabilities

F2XLLM



Input (Prompt + Source Code):

"Give me a function implemented in C with CUDA of the next subroutine implemented in Fortran and OpenACC that computes an AXPY operation:"

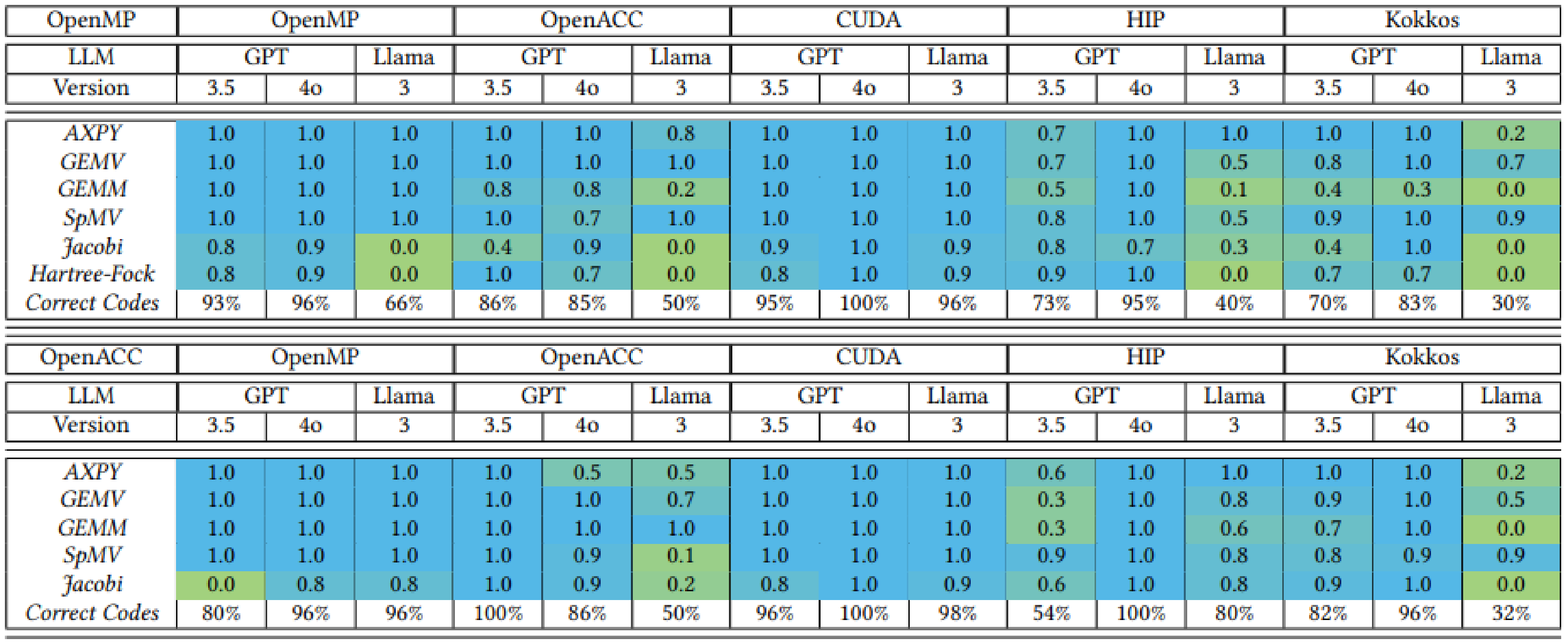
```
subroutine saxpy(n, a, x, y)
 implicit none
 integer, intent(in) :: n
 real, intent(in) :: a
 real, intent(in) :: x(n)
 real, intent(inout) :: y(n)
 integer :: i
 $acc kernels
 do i = 1, n
   y(i) = a * x(i) + y(i)
 end do
 $acc end kernels
 end subroutine saxpy
```

Output (Source Code):

```
void saxpy(int n, float a, float *x, float *y){
  float *d_x, *d_y;
  cudaMalloc(&d_x, n * sizeof(float));
  cudaMalloc(&d_y, n * sizeof(float));
  cudaMemcpy(d_x, x, n * sizeof(float), cudaMemcpyHostToDevice);
  cudaMemcpy(d_y, y, n * sizeof(float), cudaMemcpyHostToDevice);
  int blockSize = 256;
  int gridSize = (n + blockSize - 1) / blockSize;
  saxpy_parallel«<gridSize, blockSize»> (n, a, d_x, d_y);
  cudaMemcpy(y, d_y, n * sizeof(float),
  cudaMemcpyDeviceToHost);
  cudaFree(d_x);
  cudaFree(d_y);
}

__global__
void saxpy_parallel(int n, float a, float *x, float *y) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if (i < n) {
    y[i] = a * x[i] + y[i];
  }
}
```

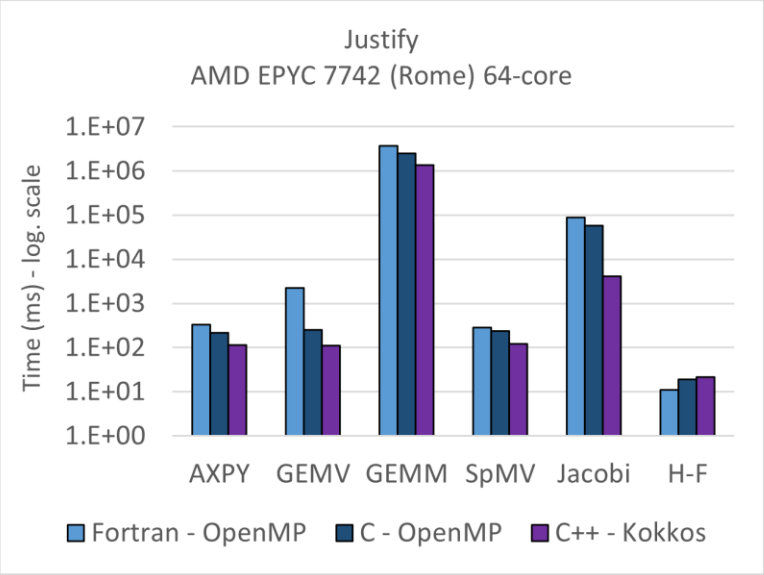
Correctness and Performance

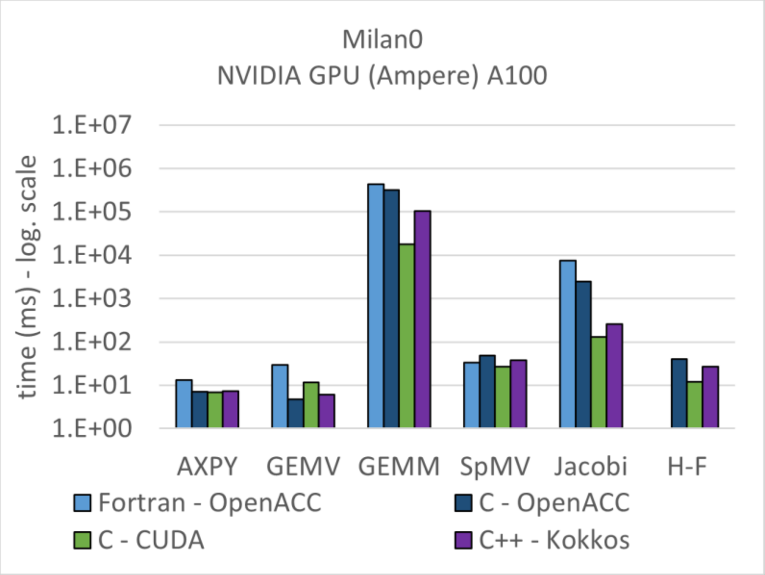
- 1,500 codes generated by AI
- Translated from 10 HPC Fortran + OpenMP and [+ OpenACC] codes
- 3 different version of LLMs
- High correctness ratios
  - GPT 3.5 -> 85.4%
  - GPT 4.o -> 94.6%
  - Llama 3 -> 67.8%
- Errors
  - Wrong use of specification (OpenMP, OpenACC, CUDA, HIP, Kokkos)
  - Wrong code
    - Parallelization of loop that shouldn't be parallelize
  - Parallelism not exploited
    - Not use of collapse, not use of proper Kokkos policy, …



OpenMP	OpenMP			OpenACC			CUDA			HIP			Kokkos		
LLM	GPT		Llama	GPT		Llama	GPT		Llama	GPT		Llama	GPT		Llama
Version	3.5	4o	3	3.5	4o	3	3.5	4o	3	3.5	4o	3	3.5	4o	3
AXPY	1.0	1.0	1.0	1.0	1.0	0.8	1.0	1.0	1.0	0.7	1.0	1.0	1.0	1.0	0.2
GEMV	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.7	1.0	0.5	0.8	1.0	0.7
GEMM	1.0	1.0	1.0	0.8	0.8	0.2	1.0	1.0	1.0	0.5	1.0	0.1	0.4	0.3	0.0
SpMV	1.0	1.0	1.0	1.0	0.7	1.0	1.0	1.0	1.0	0.8	1.0	0.5	0.9	1.0	0.9
Jacobi	0.8	0.9	0.0	0.4	0.9	0.0	0.9	1.0	0.9	0.8	0.7	0.3	0.4	1.0	0.0
Hartree-Fock	0.8	0.9	0.0	1.0	0.7	0.0	0.8	1.0	0.9	0.9	1.0	0.0	0.7	0.7	0.0
Correct Codes	93%	96%	66%	86%	85%	50%	95%	100%	96%	73%	95%	40%	70%	83%	30%

OpenACC	OpenMP			OpenACC			CUDA			HIP			Kokkos		
LLM	GPT		Llama	GPT		Llama	GPT		Llama	GPT		Llama	GPT		Llama
Version	3.5	4o	3	3.5	4o	3	3.5	4o	3	3.5	4o	3	3.5	4o	3
AXPY	1.0	1.0	1.0	1.0	0.5	0.5	1.0	1.0	1.0	0.6	1.0	1.0	1.0	1.0	0.2
GEMV	1.0	1.0	1.0	1.0	1.0	0.7	1.0	1.0	1.0	0.3	1.0	0.8	0.9	1.0	0.5
GEMM	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.3	1.0	0.6	0.7	1.0	0.0
SpMV	1.0	1.0	1.0	1.0	0.9	0.1	1.0	1.0	1.0	0.9	1.0	0.8	0.8	0.9	0.9
Jacobi	0.0	0.8	0.8	1.0	0.9	0.2	0.8	1.0	0.9	0.6	1.0	0.8	0.9	1.0	0.0
Correct Codes	80%	96%	96%	100%	86%	50%	96%	100%	98%	54%	100%	80%	82%	96%	32%





Conclusions

- Multi-modal prompting is effective for HPC programming
  - Use of codes as part of the prompts
  - Minimal prompt engineering
- Elevate considerably the quality and trustworthiness of the AI generated code
- There are still important training gaps
- Open questions
  - What is the potential benefit using fine-tuning
  - Domain-specific LLMs

ChatBLAS:
The First AI-Generated Portable BLAS Library

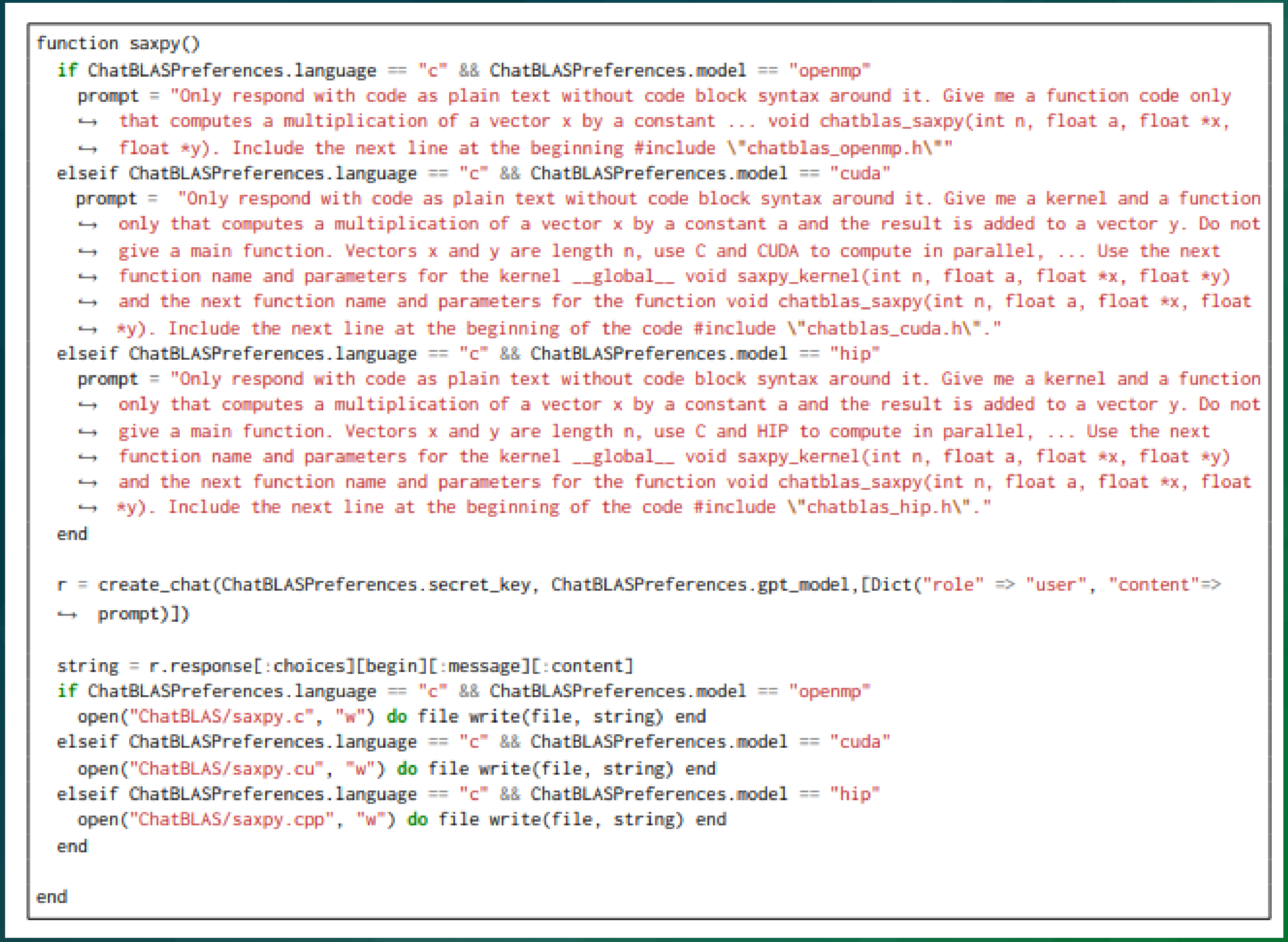
Motivation

- Evaluate the capabilities of LLMs to generate a portable and HPC library for BLAS
- Generate BLAS kernels on top of
  - OpenMP/OpenACC, CUDA, HIP, SyCL, …
- Using:
  - C/C++, Fortran, Julia, …
- Easy to validate
  - Correctness and performance
- Compilation and building

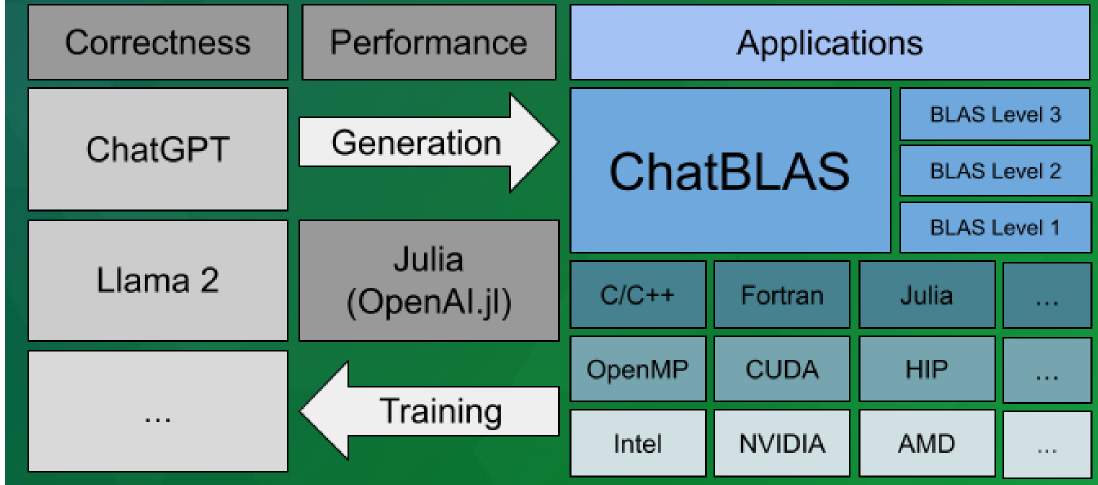
https://github.com/pedrovalerolara/ChatBLAS/tree/main

ChatBLAS

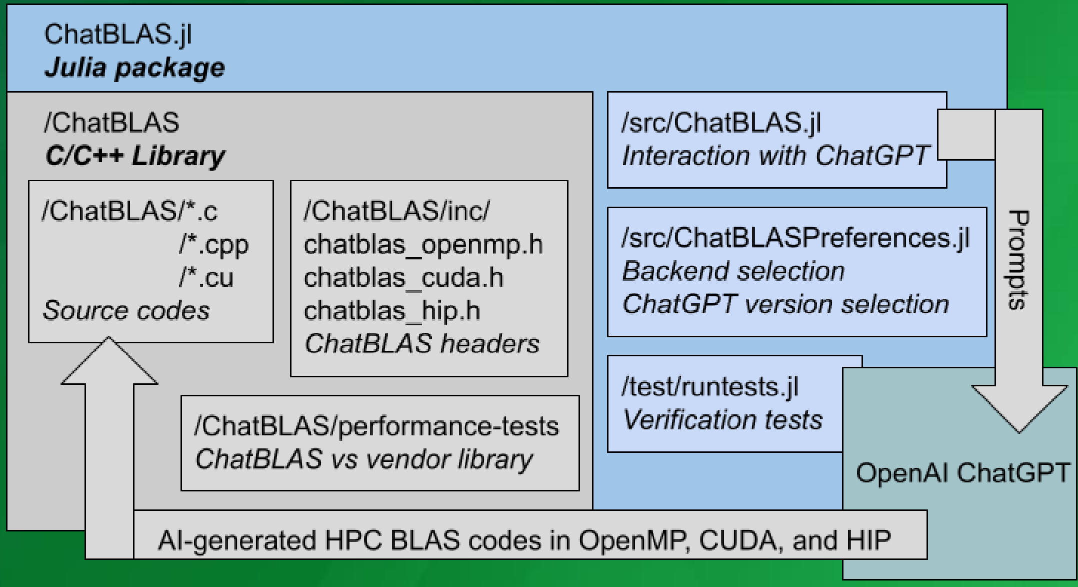
- Interact with ChatGPT to generate BLAS Kernels



```
function saxpy()
  if ChatBLASPreferences.language == "c" && ChatBLASPreferences.model == "openmp"
    prompt = "Only respond with code as plain text without code block syntax around it. Give me a function code only
    ↪  that computes a multiplication of a vector x by a constant ... void chatblas_saxpy(int n, float a, float *x,
    ↪  float *y). Include the next line at the beginning #include \"chatblas_openmp.h\""
  elseif ChatBLASPreferences.language == "c" && ChatBLASPreferences.model == "cuda"
    prompt =  "Only respond with code as plain text without code block syntax around it. Give me a kernel and a function
    ↪  only that computes a multiplication of a vector x by a constant a and the result is added to a vector y. Do not
    ↪  give a main function. Vectors x and y are length n, use C and CUDA to compute in parallel, ... Use the next
    ↪  function name and parameters for the kernel __global__ void saxpy_kernel(int n, float a, float *x, float *y)
    ↪  and the next function name and parameters for the function void chatblas_saxpy(int n, float a, float *x, float
    ↪  *y). Include the next line at the beginning of the code #include \"chatblas_cuda.h\"."
  elseif ChatBLASPreferences.language == "c" && ChatBLASPreferences.model == "hip"
    prompt = "Only respond with code as plain text without code block syntax around it. Give me a kernel and a function
    ↪  only that computes a multiplication of a vector x by a constant a and the result is added to a vector y. Do not
    ↪  give a main function. Vectors x and y are length n, use C and HIP to compute in parallel, ... Use the next
    ↪  function name and parameters for the kernel __global__ void saxpy_kernel(int n, float a, float *x, float *y)
    ↪  and the next function name and parameters for the function void chatblas_saxpy(int n, float a, float *x, float
    ↪  *y). Include the next line at the beginning of the code #include \"chatblas_hip.h\"."
  end

  r = create_chat(ChatBLASPreferences.secret_key, ChatBLASPreferences.gpt_model,[Dict("role" => "user", "content"=>
  ↪  prompt)])

  string = r.response[:choices][begin][:message][:content]
  if ChatBLASPreferences.language == "c" && ChatBLASPreferences.model == "openmp"
    open("ChatBLAS/saxpy.c", "w") do file write(file, string) end
  elseif ChatBLASPreferences.language == "c" && ChatBLASPreferences.model == "cuda"
    open("ChatBLAS/saxpy.cu", "w") do file write(file, string) end
  elseif ChatBLASPreferences.language == "c" && ChatBLASPreferences.model == "hip"
    open("ChatBLAS/saxpy.cpp", "w") do file write(file, string) end
  end

end
```



Correctness
Performance
Applications
ChatGPT
Llama 2
...
Generation
Julia (OpenAI.jl)
Training
ChatBLAS
BLAS Level 3
BLAS Level 2
BLAS Level 1
C/C++
Fortran
Julia
...
OpenMP
CUDA
HIP
...
Intel
NVIDIA
AMD
...



ChatBLAS.jl
Julia package

/ChatBLAS
C/C++ Library

/ChatBLAS/*.c
/*.cpp
/*.cu
Source codes

/ChatBLAS/inc/
chatblas_openmp.h
chatblas_cuda.h
chatblas_hip.h
ChatBLAS headers

/ChatBLAS/performance-tests
ChatBLAS vs vendor library

/src/ChatBLAS.jl
Interaction with ChatGPT

/src/ChatBLASPreferences.jl
Backend selection
ChatGPT version selection

/test/runtests.jl
Verification tests

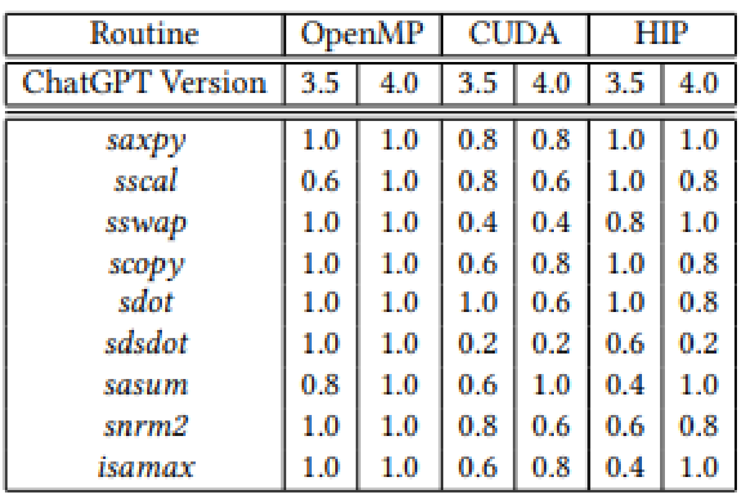
Prompts

OpenAI ChatGPT

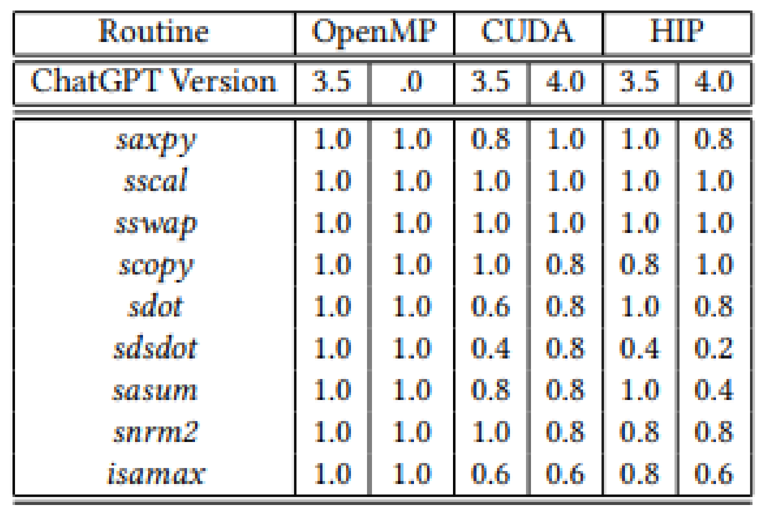
AI-generated HPC BLAS codes in OpenMP, CUDA, and HIP

Correctness

- Prompt
  - *Give me a function code only that computes a multiplication of a vector x by a constant a and the result is added to a vector y. Vectors x and y are length n, use C and CUDA to compute in parallel include the next line in the code, and use the next function name and parameters void chatblas_saxpy(int n, float a, float *x, float *y). Include the next line at the beginning #include chatblas_cuda.h*
- Prompt Engineering
  - *Give me a kernel and a function only that computes a multiplication of a vector x by a constant a and the result is added to a vector y. Do not give a main function. Vectors x and y are length n, use C and CUDA to compute in parallel, allocate and free the GPU vectors, and make the CPU - GPU memory transfers in the function. The size of blocks of threads and the number of blocks must be defined. Use the next function name and parameters for the kernel __global__ void saxpy_kernel(int n, float a, float *x, float *y), and the next function name and parameters for the function void chatblas_saxpy(int n, float a, float *x, float *y). Include the next line at the beginning of the code #include "chatblas_cuda.h"*
  - Correctness was elevated about 20%



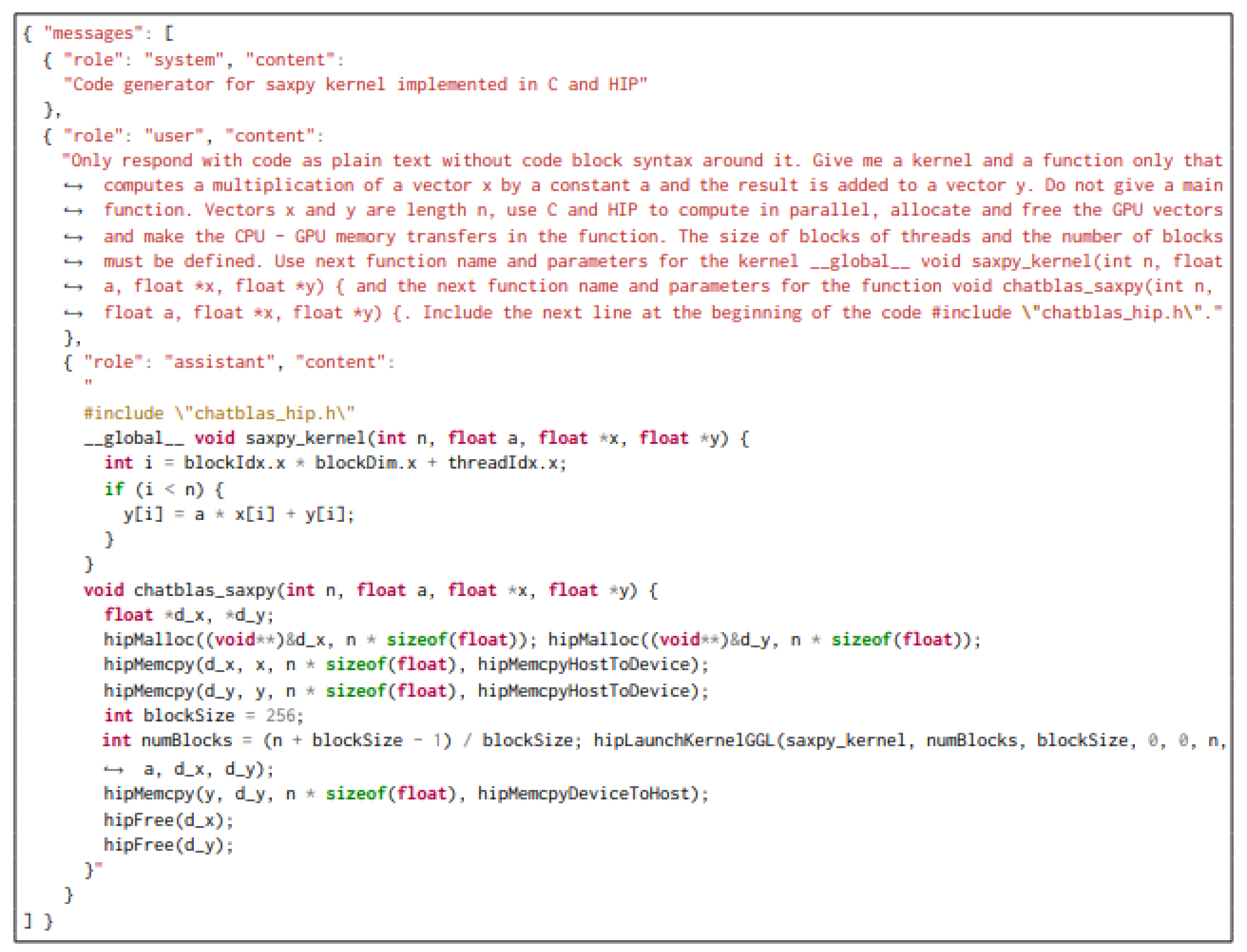
Routine	OpenMP		CUDA		HIP	
ChatGPT Version	3.5	4.0	3.5	4.0	3.5	4.0
saxpy	1.0	1.0	0.8	0.8	1.0	1.0
sscal	0.6	1.0	0.8	0.6	1.0	0.8
sswap	1.0	1.0	0.4	0.4	0.8	1.0
scopy	1.0	1.0	0.6	0.8	1.0	0.8
sdot	1.0	1.0	1.0	0.6	1.0	0.8
sdsdot	1.0	1.0	0.2	0.2	0.6	0.2
sasum	0.8	1.0	0.6	1.0	0.4	1.0
snrm2	1.0	1.0	0.8	0.6	0.6	0.8
isamax	1.0	1.0	0.6	0.8	0.4	1.0



Routine	OpenMP		CUDA		HIP	
ChatGPT Version	3.5	.0	3.5	4.0	3.5	4.0
saxpy	1.0	1.0	0.8	1.0	1.0	0.8
sscal	1.0	1.0	1.0	1.0	1.0	1.0
sswap	1.0	1.0	1.0	1.0	1.0	1.0
scopy	1.0	1.0	1.0	0.8	0.8	1.0
sdot	1.0	1.0	0.6	0.8	1.0	0.8
sdsdot	1.0	1.0	0.4	0.8	0.4	0.2
sasum	1.0	1.0	0.8	0.8	1.0	0.4
snrm2	1.0	1.0	1.0	0.8	0.8	0.8
isamax	1.0	1.0	0.6	0.6	0.8	0.6

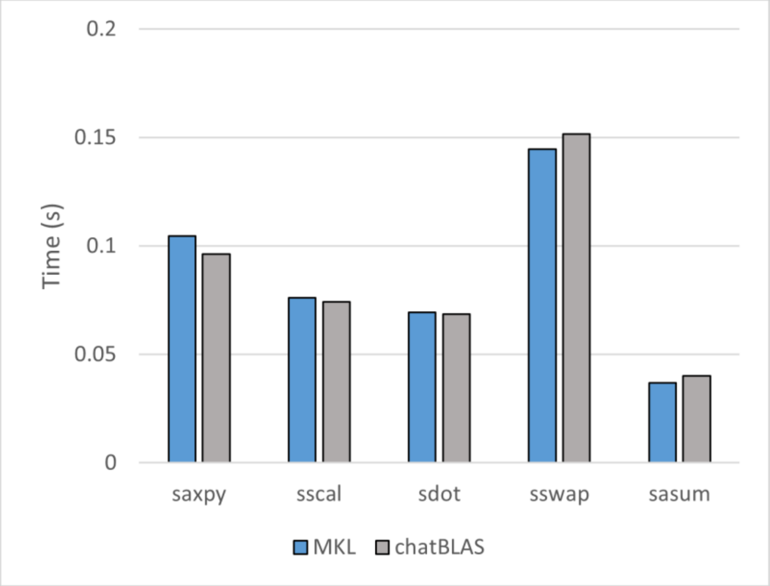
Fine-Tuning for Correctness

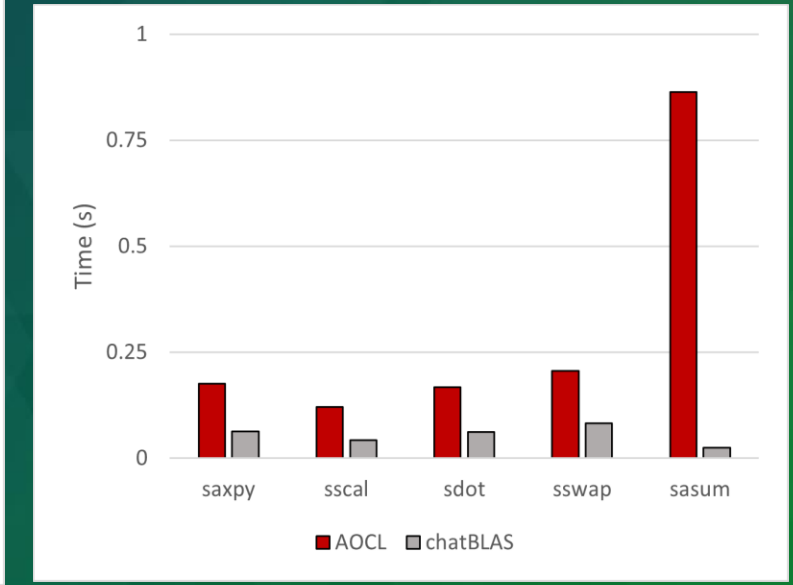
- Correctness level were elevated to 100% of the cases using fine-tuning

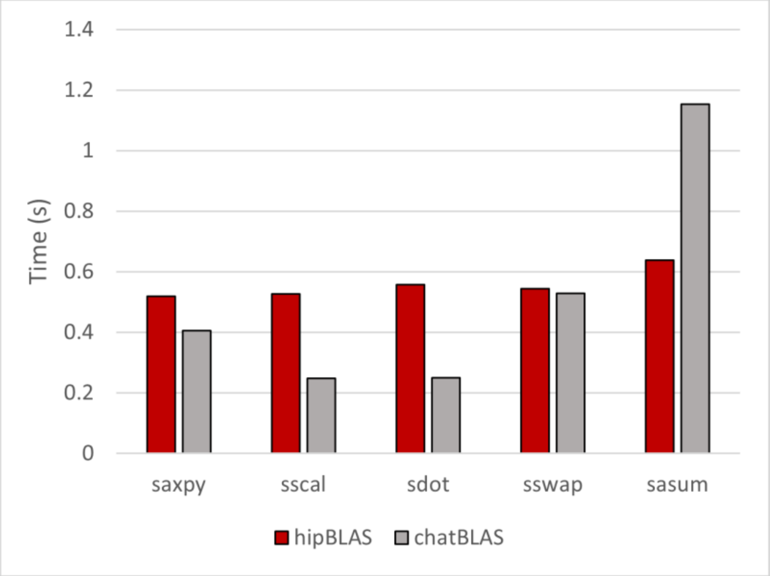


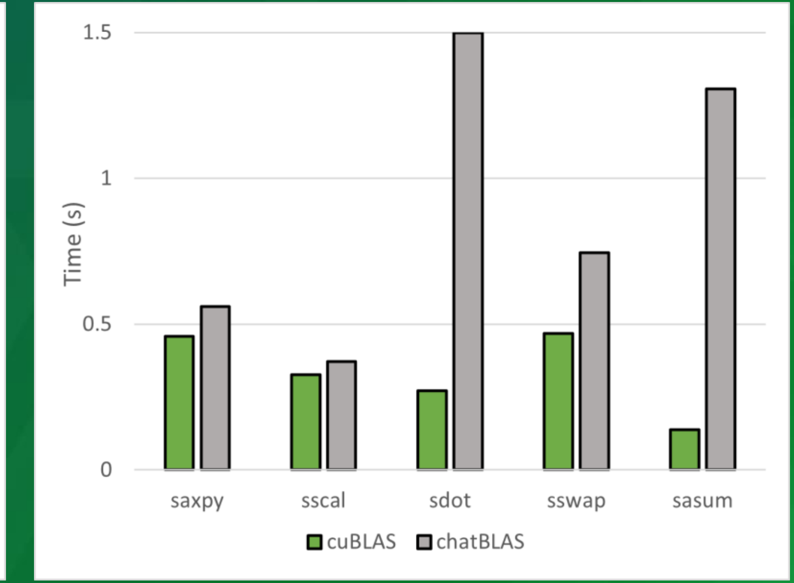
```
{ "messages": [
  { "role": "system", "content":
    "Code generator for saxpy kernel implemented in C and HIP"
  },
  { "role": "user", "content":
    "Only respond with code as plain text without code block syntax around it. Give me a kernel and a function only that
    ↪  computes a multiplication of a vector x by a constant a and the result is added to a vector y. Do not give a main
    ↪  function. Vectors x and y are length n, use C and HIP to compute in parallel, allocate and free the GPU vectors
    ↪  and make the CPU - GPU memory transfers in the function. The size of blocks of threads and the number of blocks
    ↪  must be defined. Use next function name and parameters for the kernel __global__ void saxpy_kernel(int n, float
    ↪  a, float *x, float *y) { and the next function name and parameters for the function void chatblas_saxpy(int n,
    ↪  float a, float *x, float *y) {. Include the next line at the beginning of the code #include \"chatblas_hip.h\"."
    },
    { "role": "assistant", "content":
      "

      #include \"chatblas_hip.h\"
      __global__ void saxpy_kernel(int n, float a, float *x, float *y) {
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        if (i < n) {
          y[i] = a * x[i] + y[i];
        }
      }
      void chatblas_saxpy(int n, float a, float *x, float *y) {
        float *d_x, *d_y;
        hipMalloc((void**)&d_x, n * sizeof(float)); hipMalloc((void**)&d_y, n * sizeof(float));
        hipMemcpy(d_x, x, n * sizeof(float), hipMemcpyHostToDevice);
        hipMemcpy(d_y, y, n * sizeof(float), hipMemcpyHostToDevice);
        int blockSize = 256;
        int numBlocks = (n + blockSize - 1) / blockSize; hipLaunchKernelGGL(saxpy_kernel, numBlocks, blockSize, 0, 0, n,
        ↪  a, d_x, d_y);
        hipMemcpy(y, d_y, n * sizeof(float), hipMemcpyDeviceToHost);
        hipFree(d_x);
        hipFree(d_y);
      }"
    }
] }
```

Performance

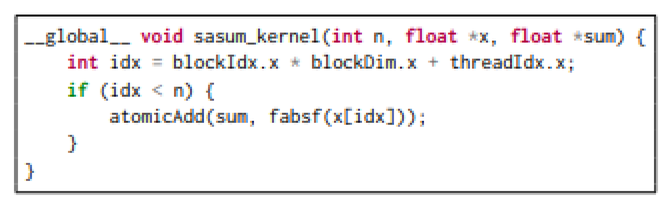




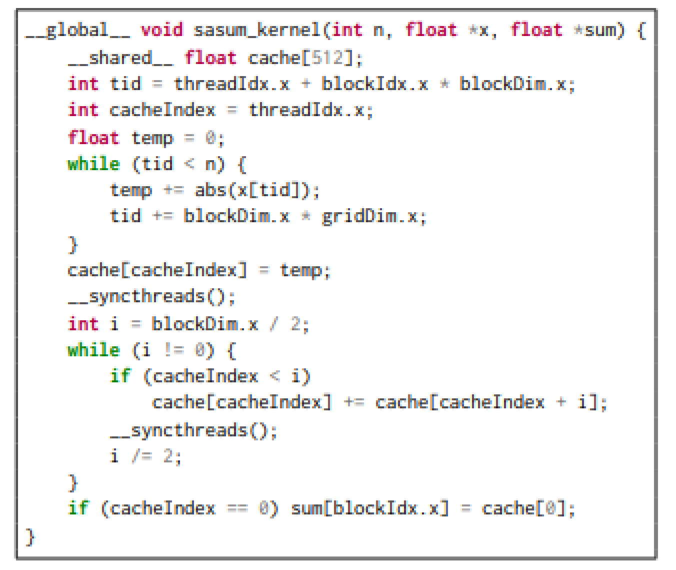




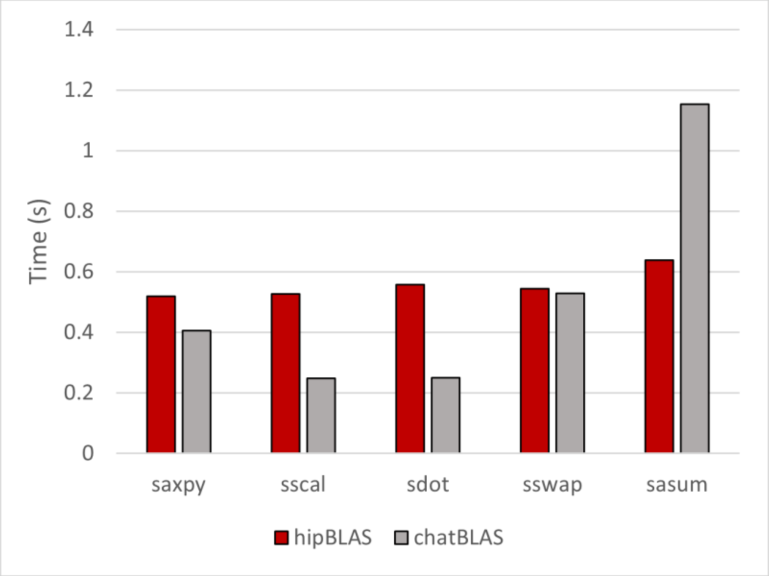
Fine-Tuning for Performance

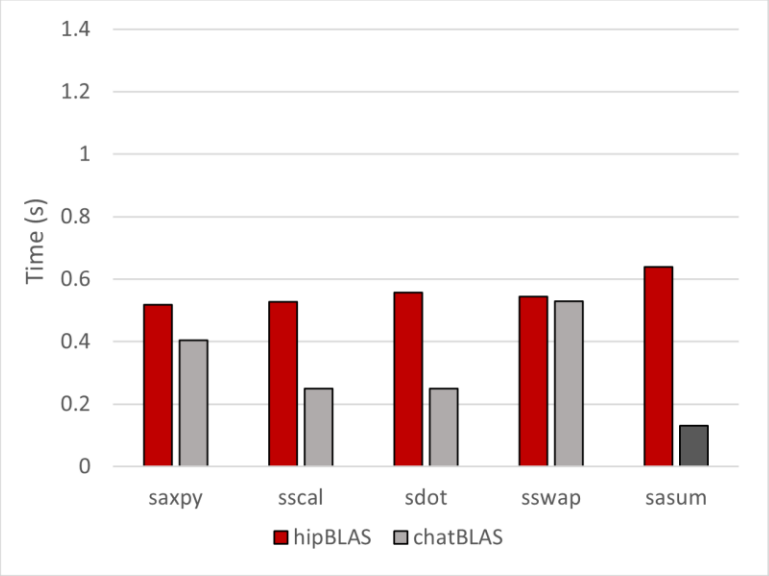


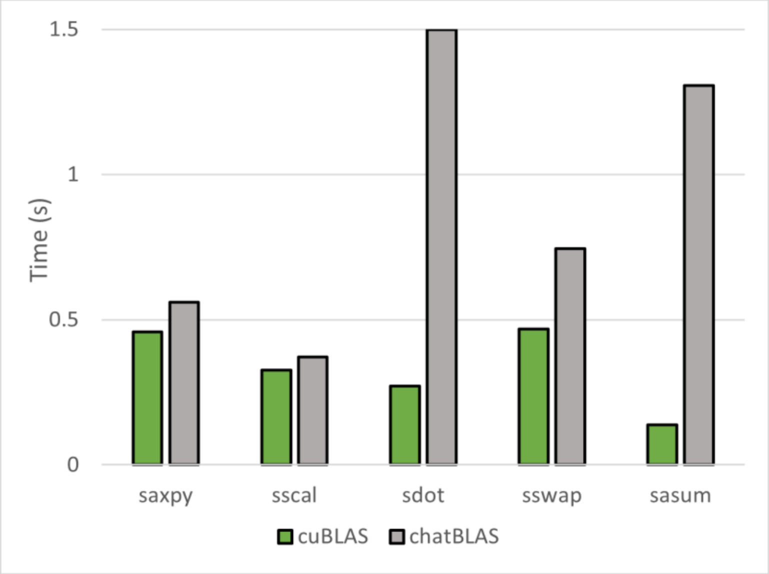
```
__global__ void sasum_kernel(int n, float *x, float *sum) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < n) {
        atomicAdd(sum, fabsf(x[idx]));
    }
}
```

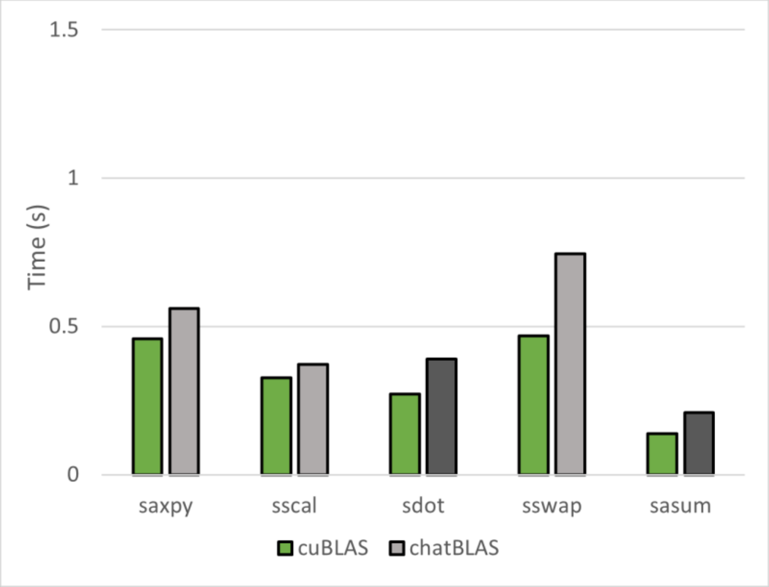


```
__global__ void sasum_kernel(int n, float *x, float *sum) {
    __shared__ float cache[512];
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    int cacheIndex = threadIdx.x;
    float temp = 0;
    while (tid < n) {
        temp += abs(x[tid]);
        tid += blockDim.x * gridDim.x;
    }
    cache[cacheIndex] = temp;
    __syncthreads();
    int i = blockDim.x / 2;
    while (i != 0) {
        if (cacheIndex < i)
            cache[cacheIndex] += cache[cacheIndex + i];
        __syncthreads();
        i /= 2;
    }
    if (cacheIndex == 0) sum[blockIdx.x] = cache[0];
}
```









Conclusions

- AI can generate optimized BLAS codes for different targets
  - Filling the gaps of vendor libraries
  - Providing competitive or even better performance than vendor libraries
- Prompt engineering can improve the correctness of the codes, but still does not guarantee 100% correctness
- Fine-tuning can help to
  - elevate levels of correctness up to 100%
  - elevate performance on codes considerably (up to 2x better performance)

ChatHPC: What is that?

Building the foundations towards an AI-Assisted and high-productive HPC ecosystem

In a very fragmented and more and more complex and demanding HPC ecosystem composed of many and relatively small tools/efforts our approach is:

Many small AI agents but highly optimized for one target,

instead of one huge and general LLM

ChatHPC: What is the motivation?

- Not all the data is on the Internet
- The responses are not precise, are incomplete, or are not correct
  - ***Even when the data come from the Internet, big LLMs***
- HPC is a science, LLMs need to be Precise, Complete, and Correct
- ***Can the fine-tuned small (7B) LLMs generate as good information or even better than the big (1T) LLMs?***
  - 3 orders of magnitude less data, faster, and less power-consumption

ChatHPC: What is our approach?

- We focus on fine-tuning, instead of training:
  - Fine-tuning -> fast and cheap (ChatKokkos ~ 20 min)
  - Training -> slow and expensive
- Collaborate with Industry (cost effective)
  - CodeLlama, oLlama, hugging face, …
- Scalable:
  - Fine tuning process is the same
  - Data and evaluation is different

ChatHPC: How to fine tune?

The better the data the better the model

It is more about quality than quantity!

How to organize the data?

- *Using contexts learning (ChatKokkos)*
  - *Introduction to Kokkos programming model*
  - *Kokkos installation*
  - *Kokkos development*
  - *Automatic parallelization (e.g., sequential to Kokkos)*
  - *Automatic translation (e.g., from OpenMP to Kokkos)*

```
[ {
   "question": "What is Kokkos?",
   "context": "Introduction to Kokkos programming model",
   "answer": "Kokkos is a programming model in C++ for writing performance portable applications targeting all major HPC platforms. For that purpose, it provides abstractions for both parallel execution of code and data management. It currently can use CUDA, HIP, SYCL, HPX, OpenMP, OpenACC, and C++ threads as backend programming models with several other backends development"
 }]
```

ChatHPC: Infrastructure?

- ORNL Experimental Computing Laboratory
  - https://www.excl.ornl.gov/
- Same platform
  - https://code.ornl.gov/ChatHPC/ChatHPC-project
  - Pytorch, ipython, vscode, CodeLlama, oLlama, …
  - Disc
- Hackathons
- Documentation, videos, slides, …

ChatHPC: First Prototype Why Julia??

- ***Think in Python, but now imagine that it works well on HPC***
- A JIT language on top of LLVM
  - Easy-to-use and agile interface with C performance
- Julia syntax is optimized for Science
- Native syntax (no manual implemented wrappers of wrapper of …) for HPC support
  - Threads, CUDA, AMDGPU, OneAPI, MPI, DAGGER, etc.
- Native support for AI -> HPC-AI integration
- Integrated and efficient support for packaging, reproducibility, CI/CD, …
- All this makes Julia an ecosystem motivated by performance and productivity

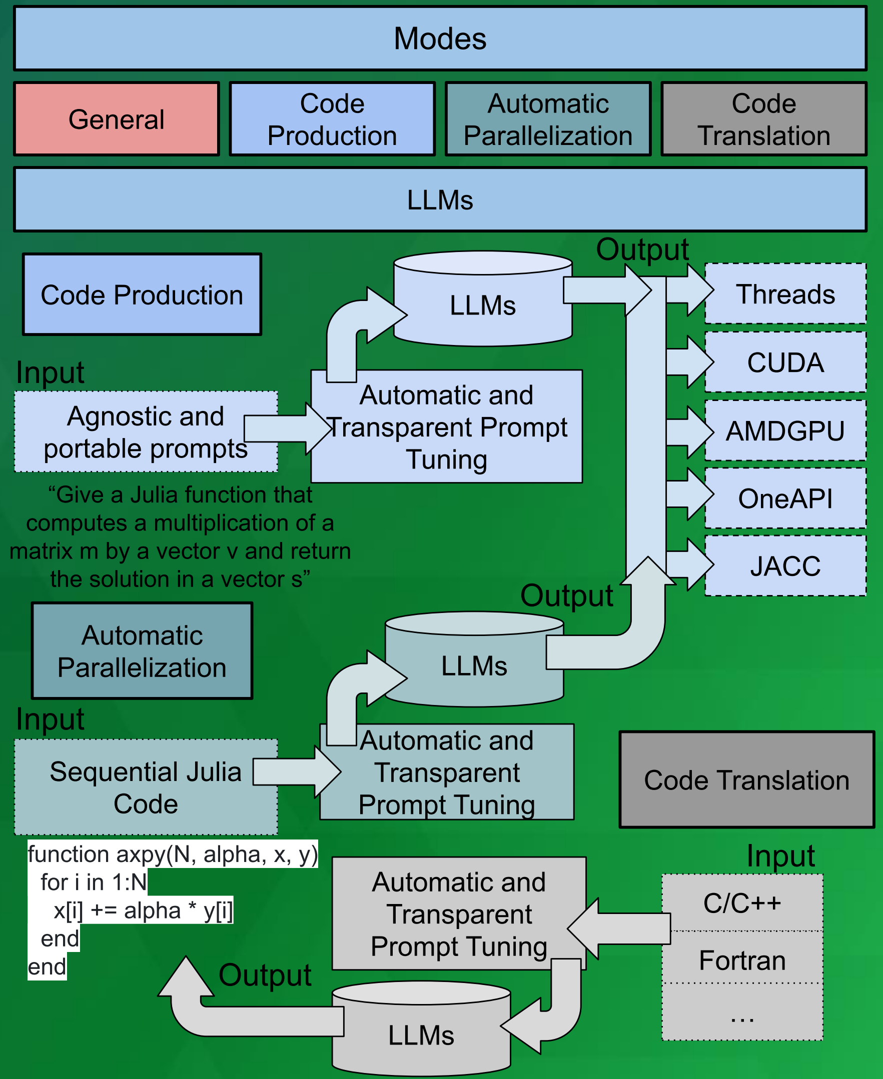
https://julialang.org/

ChatHPC Architecture

Assist HPC and Scientific code Julia developers

- Integrated in the real-time and interactive read-eval-print loop (REPL) Just-in-time and LLVM-based Julia ecosystem
- Interact with LLMs for HPC targets (modes of use):
  - Code/Kernel generation
  - Automatic parallelization
    - Users provide a sequential code and LLM provides a parallel version of the sequential code
  - Code translation
    - C/C++, Fortran, …, code provided and translated to Julia code
- Transparent (portable and auto) prompt-engineering
  - Same prompt can be used for different targets
- Supported models
  - Threads, AMDGPU, CUDA, and OneAPI

ChatHPC


Modes
General
Code Production
Automatic Parallelization
Code Translation
LLMs
Code Production
Input
Agnostic and portable prompts
Automatic and Transparent Prompt Tuning
LLMs
Output
Threads
CUDA
AMDGPU
OneAPI
JACC
Automatic Parallelization
Input
Sequential Julia Code
Automatic and Transparent Prompt Tuning
LLMs
Output
Code Translation
Input
C/C++
Fortran
…
Automatic and Transparent Prompt Tuning
LLMs
Output

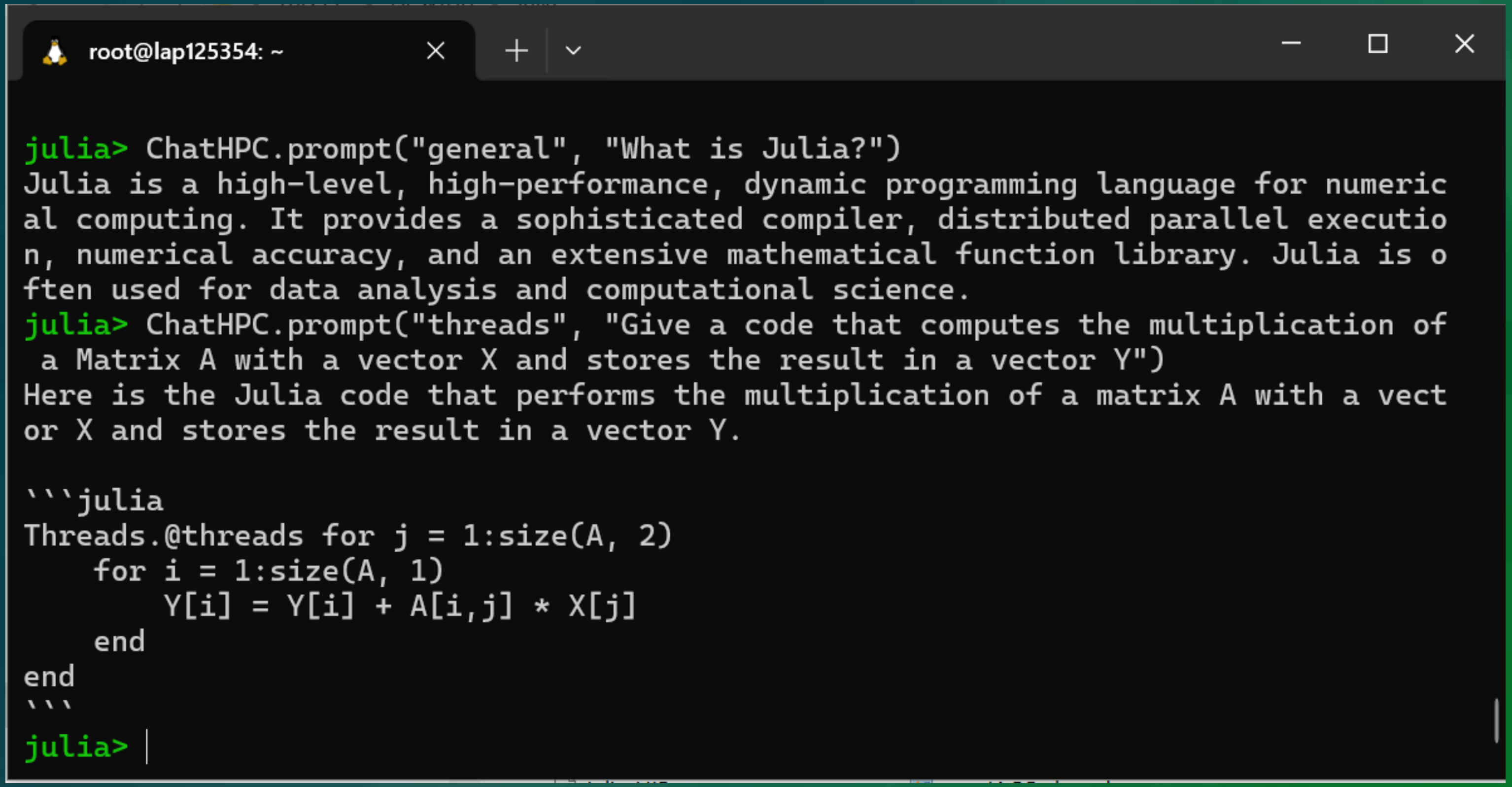
"Give a Julia function that computes a multiplication of a matrix m by a vector v and return the solution in a vector s"

```
function axpy(N, alpha, x, y)
 for i in 1:N
  x[i] += alpha * y[i]
 end
end
```

Code/Kernel Generation (Threads)

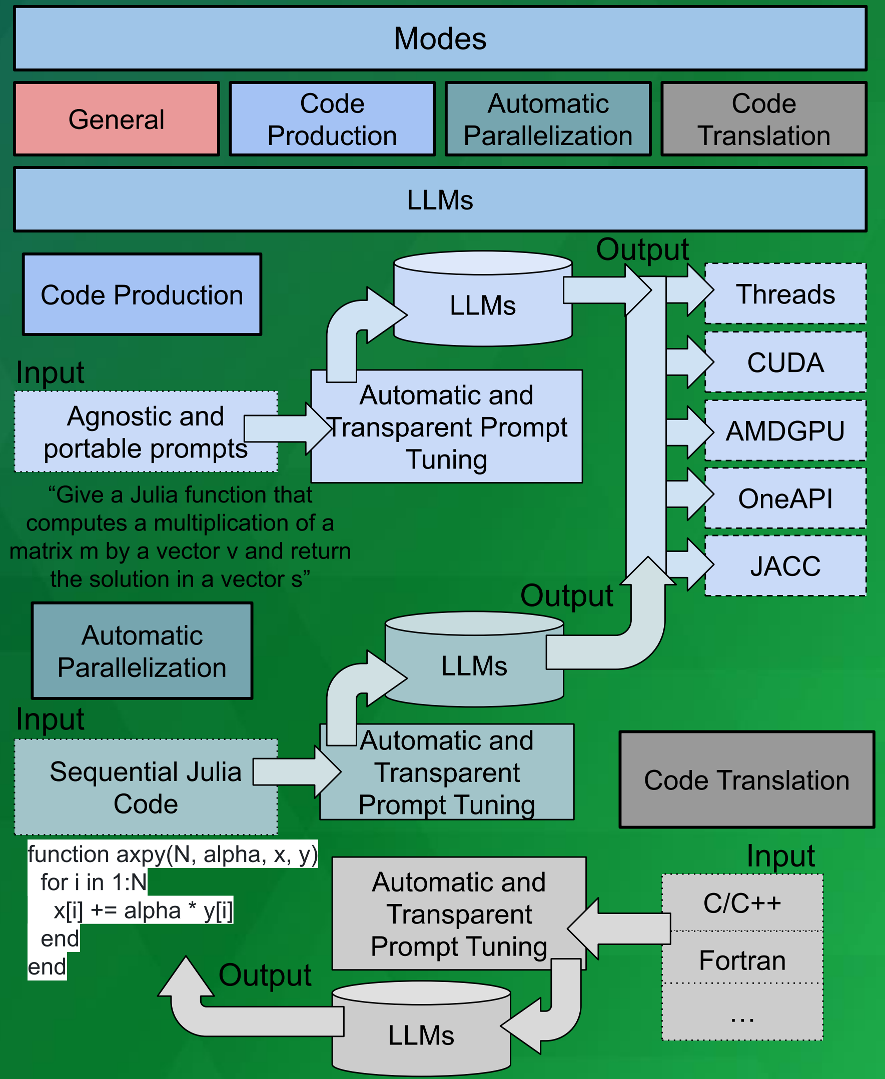
Code Generation

- Hidden details (prompt-tuning) about granularity, memory management, synchronization, optimization (shared memory), etc.
- Functions



~~~
julia> ChatHPC.prompt("general", "What is Julia?")
Julia is a high-level, high-performance, dynamic programming language for numeric
al computing. It provides a sophisticated compiler, distributed parallel executio
n, numerical accuracy, and an extensive mathematical function library. Julia is o
ften used for data analysis and computational science.
julia> ChatHPC.prompt("threads", "Give a code that computes the multiplication of
 a Matrix A with a vector X and stores the result in a vector Y")
Here is the Julia code that performs the multiplication of a matrix A with a vect
or X and stores the result in a vector Y.

```julia
Threads.@threads for j = 1:size(A, 2)
    for i = 1:size(A, 1)
        Y[i] = Y[i] + A[i,j] * X[j]
    end
end
```
julia> 
~~~

ChatHPC



Modes

General | Code Production | Automatic Parallelization | Code Translation

LLMs

Code Production

Input: Agnostic and portable prompts → Automatic and Transparent Prompt Tuning → LLMs → Output: Threads, CUDA, AMDGPU, OneAPI, JACC

"Give a Julia function that computes a multiplication of a matrix m by a vector v and return the solution in a vector s"

Automatic Parallelization

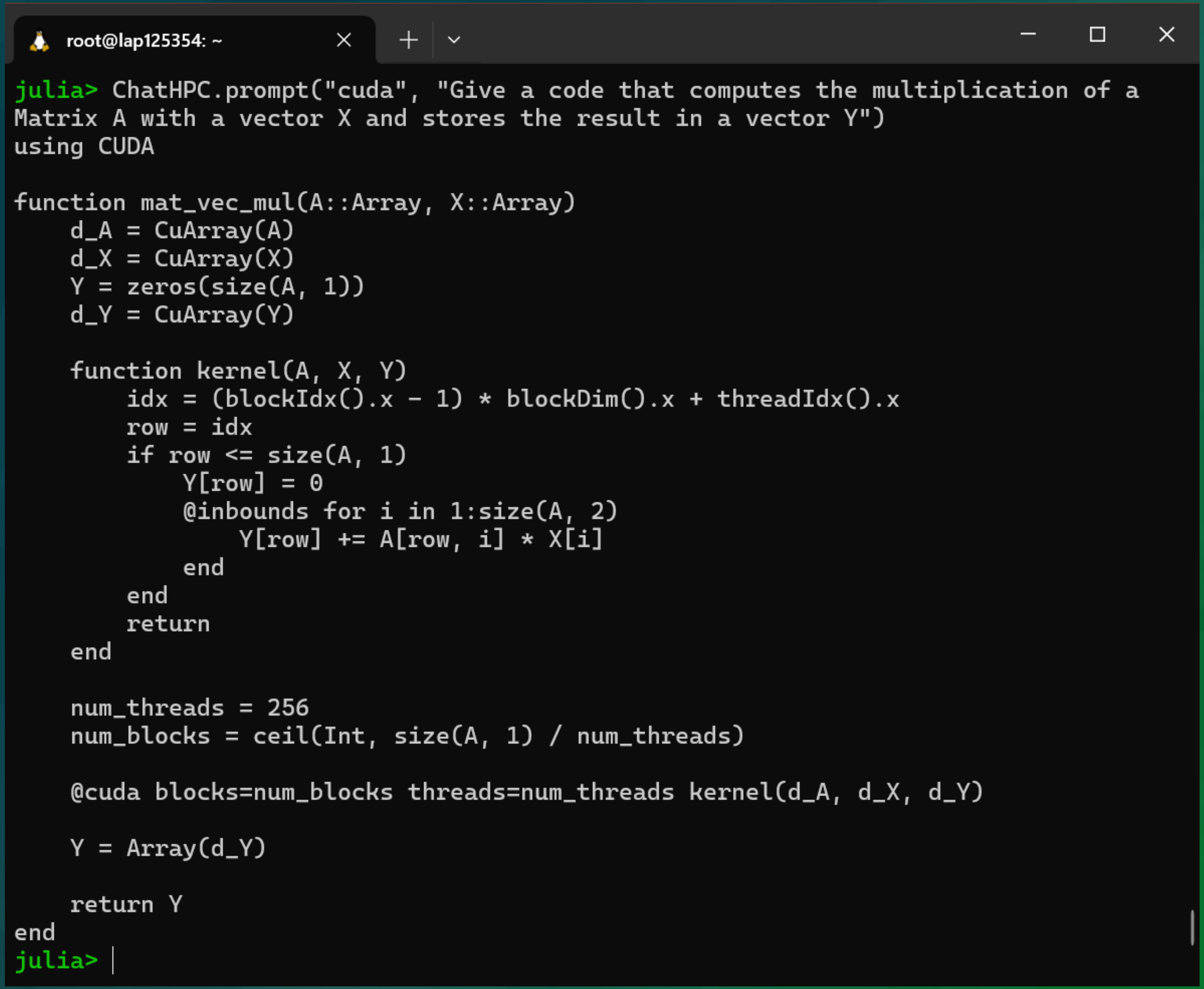
Input: Sequential Julia Code → Automatic and Transparent Prompt Tuning → LLMs → Output

```
function axpy(N, alpha, x, y)
  for i in 1:N
    x[i] += alpha * y[i]
  end
end
```

Code Translation

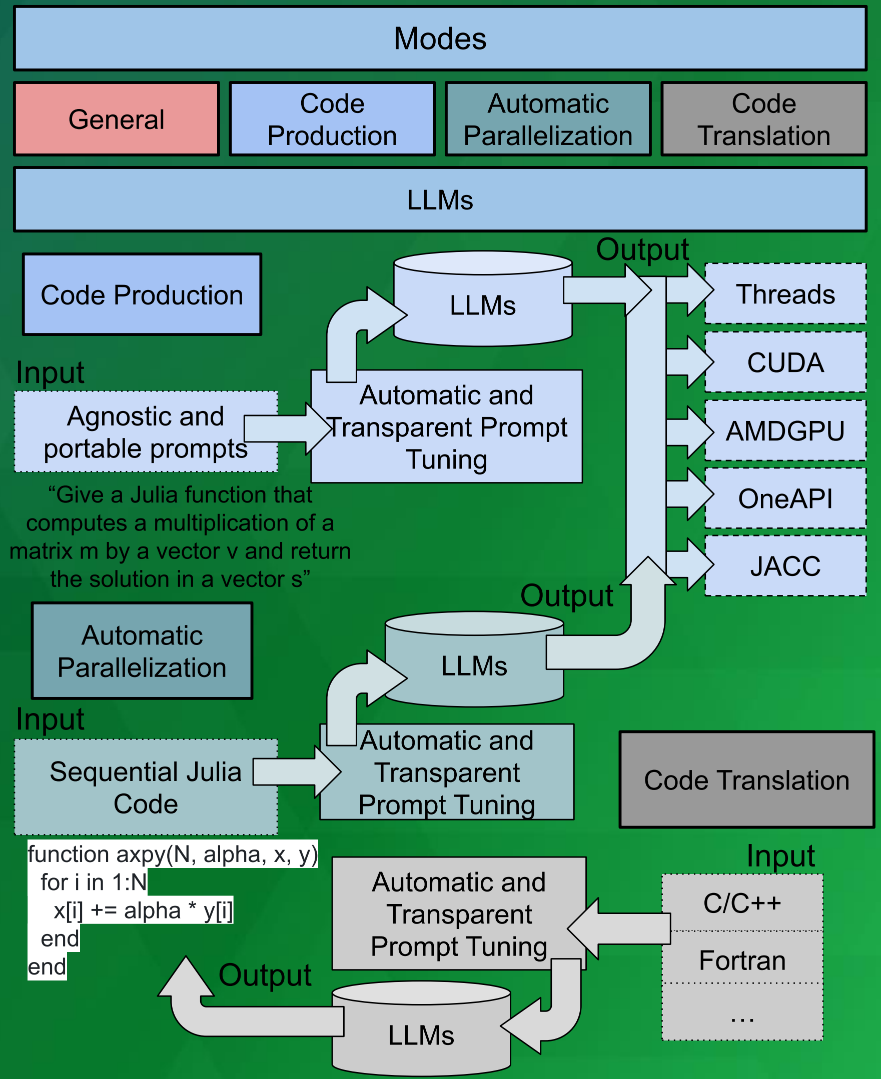
Input: C/C++, Fortran, … → Automatic and Transparent Prompt Tuning → LLMs → Output

Code/Kernel Generation (CUDA)



```
julia> ChatHPC.prompt("cuda", "Give a code that computes the multiplication of a
Matrix A with a vector X and stores the result in a vector Y")
using CUDA

function mat_vec_mul(A::Array, X::Array)
    d_A = CuArray(A)
    d_X = CuArray(X)
    Y = zeros(size(A, 1))
    d_Y = CuArray(Y)

    function kernel(A, X, Y)
        idx = (blockIdx().x - 1) * blockDim().x + threadIdx().x
        row = idx
        if row <= size(A, 1)
            Y[row] = 0
            @inbounds for i in 1:size(A, 2)
                Y[row] += A[row, i] * X[i]
            end
        end
        return
    end

    num_threads = 256
    num_blocks = ceil(Int, size(A, 1) / num_threads)

    @cuda blocks=num_blocks threads=num_threads kernel(d_A, d_X, d_Y)

    Y = Array(d_Y)

    return Y
end
julia>
```

ChatHPC



Modes

General | Code Production | Automatic Parallelization | Code Translation

LLMs

Code Production

Input

Agnostic and portable prompts

“Give a Julia function that computes a multiplication of a matrix m by a vector v and return the solution in a vector s”

Automatic and Transparent Prompt Tuning

LLMs

Output

Threads

CUDA

AMDGPU

OneAPI

JACC

Automatic Parallelization

Input

Sequential Julia Code

```
function axpy(N, alpha, x, y)
  for i in 1:N
    x[i] += alpha * y[i]
  end
end
```

Automatic and Transparent Prompt Tuning

LLMs

Output

Code Translation

Input

C/C++

Fortran

…

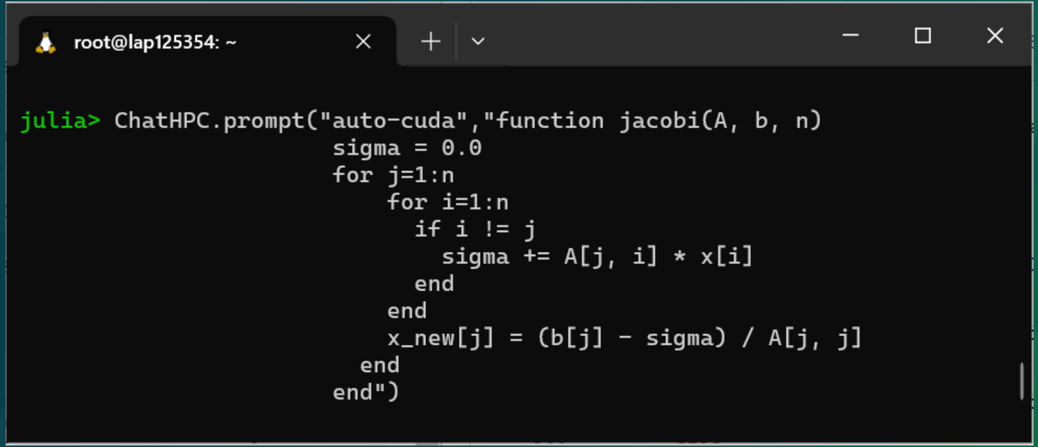
Automatic and Transparent Prompt Tuning

LLMs

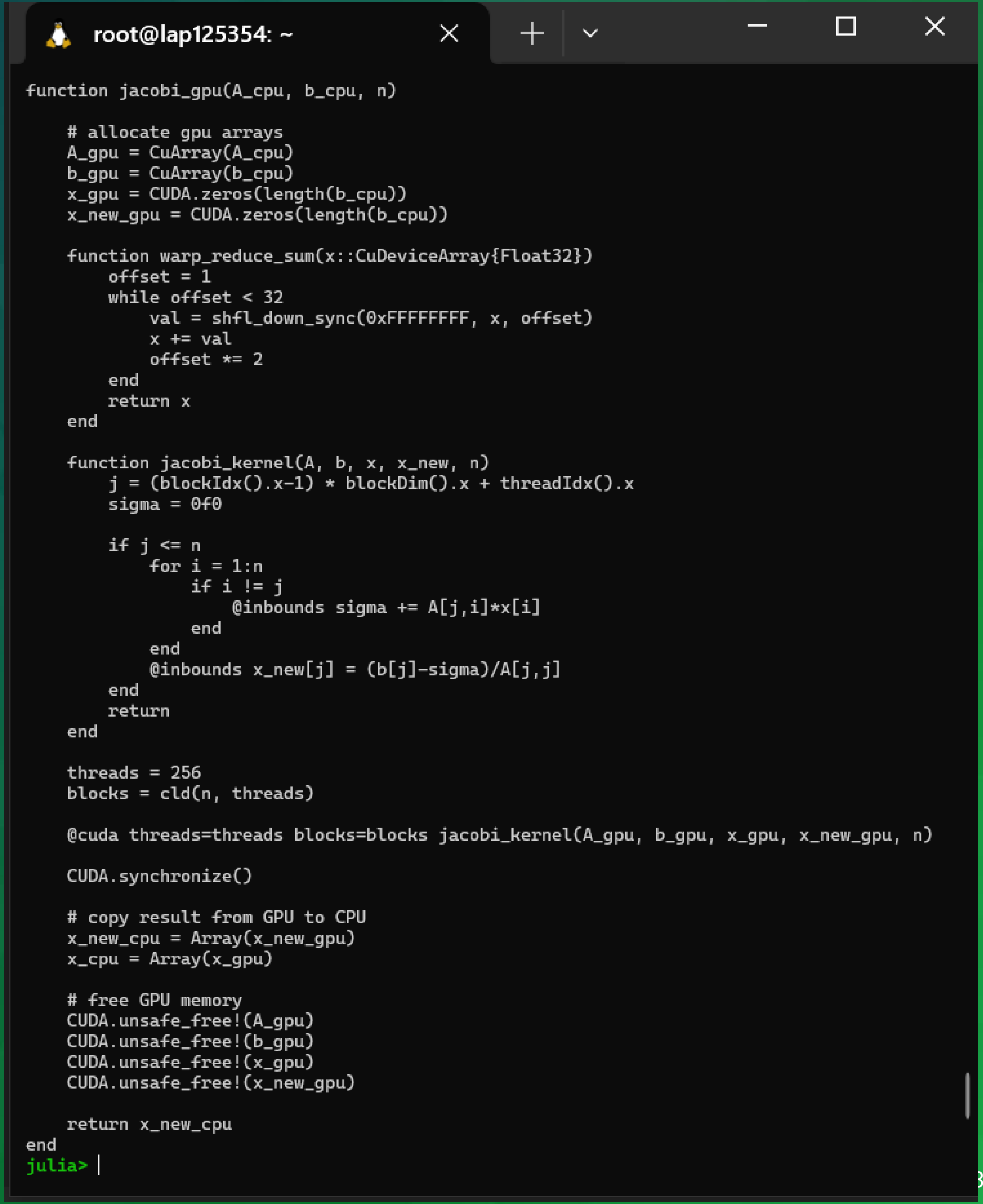
Output

Automatic Parallelization

- Do you see anything weird in the code?
  - The function warp_reduce_sum is not necessary
  - The code is functional despite this



```
julia> ChatHPC.prompt("auto-cuda","function jacobi(A, b, n)
                       sigma = 0.0
                       for j=1:n
                           for i=1:n
                             if i != j
                               sigma += A[j, i] * x[i]
                             end
                           end
                           x_new[j] = (b[j] - sigma) / A[j, j]
                         end
                       end")
```



```
function jacobi_gpu(A_cpu, b_cpu, n)

    # allocate gpu arrays
    A_gpu = CuArray(A_cpu)
    b_gpu = CuArray(b_cpu)
    x_gpu = CUDA.zeros(length(b_cpu))
    x_new_gpu = CUDA.zeros(length(b_cpu))

    function warp_reduce_sum(x::CuDeviceArray{Float32})
        offset = 1
        while offset < 32
            val = shfl_down_sync(0xFFFFFFFF, x, offset)
            x += val
            offset *= 2
        end
        return x
    end

    function jacobi_kernel(A, b, x, x_new, n)
        j = (blockIdx().x-1) * blockDim().x + threadIdx().x
        sigma = 0f0

        if j <= n
            for i = 1:n
                if i != j
                    @inbounds sigma += A[j,i]*x[i]
                end
            end
            @inbounds x_new[j] = (b[j]-sigma)/A[j,j]
        end
        return
    end

    threads = 256
    blocks = cld(n, threads)

    @cuda threads=threads blocks=blocks jacobi_kernel(A_gpu, b_gpu, x_gpu, x_new_gpu, n)

    CUDA.synchronize()

    # copy result from GPU to CPU
    x_new_cpu = Array(x_new_gpu)
    x_cpu = Array(x_gpu)

    # free GPU memory
    CUDA.unsafe_free!(A_gpu)
    CUDA.unsafe_free!(b_gpu)
    CUDA.unsafe_free!(x_gpu)
    CUDA.unsafe_free!(x_new_gpu)

    return x_new_cpu
end
julia> |
```

ChatHPC: Community

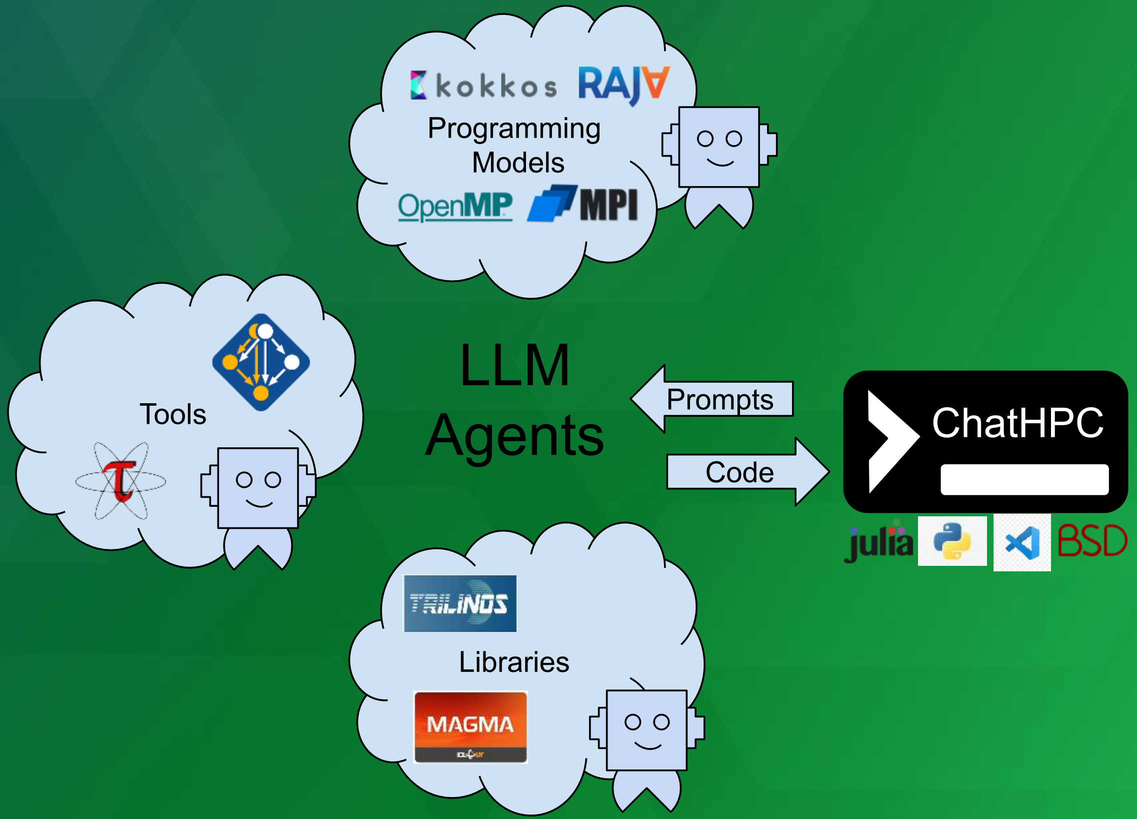
- Multiple efforts
  - Programming models (ChatKokkos, ChatIRIS, ChatOpenMP, …)
  - Math libraries (ChatPlasma, ChatBLAS, …)
  - Packaging (ChatSpack)
  - Tooling (ChatTau)
  - Autotuning
- International, multi-institution, and public-private collaboration
  - ORNL, SNL, Emory University, TU Dresden, Appentra, University of Delaware, Carnegie Mellon University, MIT, …

ChatHPC: What is the plan?

ChatHPC as a collection of LLM (AI agents) for specific HPC tools and efforts

HPC stack	ChatHPC
Applications	ChatVASP, ChatQMCPACK, …
Math Libraries	ChatBLAS, ChatPlasma, ChatMAGMA?, …
Programming Models	ChatKokkos, ChatIRIS, …
Tools	ChatTau, ChatPapi
Packaging	ChatSpack
I/O	ChatADIOS2

HPC efforts	ChatHPC
Automatic Parallelization	ChatKokkos
Automatic Translation	ChatKokkos, ChatPlasma
Installation	ChatSpack, ChatKokkos, …
Autotuning	ChatTau, ChatPapi
V&V	



References

- ***William F. Godoy***, Pedro Valero-Lara, Keita Teranishi, Prasanna Balaprakash, Jeffrey S. Vetter: **Evaluation of OpenAI Codex for HPC Parallel Programming Models Kernel Generation.** ICPP Workshops 2023:

- *Pedro Valero-Lara*, Alexis Huante, Mustafa Al Lail, William F. Godoy, Keita Teranishi, Prasanna Balaprakash, Jeffrey S. Vetter: ***Comparing Llama-2 and GPT-3 LLMs for HPC kernels generation***. LCPC. 2023.

- *William F. Godoy*, Pedro Valero-Lara, Keita Teranishi, Prasanna Balaprakash, Jeffrey S. Vetter: **Large language model evaluation for high-performance computing software development.** Concurr. Comput. Pract. Exp. 36(26) (2024)