ChatHPC: Building the Foundation for a Productive and Trustworthy AI-Assisted HPC Ecosystem

SCA2026 Supercomputing Asia — HPC Asia 2026

Pedro Valero-Lara, *valerolarap@ornl.gov* | Aaron Young, *younger@ornl.gov* | William F. Godoy, *godoywf@ornl.gov*
Keita Teranishi, *teranishik@ornl.gov* | Jeffrey S. Vetter, *vetter@ornl.gov*

Oak Ridge National Laboratory — U.S. Department of Energy

Motivation

Democratize LLMs for HPC community

- Creation of reliable, highly specialized and optimized AI assistants across major HPC components: Programing Models, I/O, Math Libraries, Tooling, and beyond
- Production of high-quality software with level of trustworthiness of up to 90% higher than OpenAI ChatGPT-4o

A Reliable and Efficient Ecosystem

Based on robust and open-source tools

- Code Llama 7B parameter fine-tuning, LoRA, and Pytorch
- Low computational requirements
- Code Llama (13GB), AI-Assistant (~100MB), training/testing data (~KB)
- The creation of AI-assistants takes in order of a few minutes (< 15 min) on 2x NVIDIA GPUs

ChatHPC Library

Accelerating AI assistants (LLMs) production for HPC

- An easy-to-use front-end Python library for AI assistants' creation and testing focus on data quality

ChatHPC CLI:

```
$ chathpc train   # Finetune the assistent.
$ chathpc verify  # Verify the assistent on training set.
$ chathpc test    # Test the assistent on unseen data.
$ chathpc run     # Interactivly run the assistent.
```

Interactive run session:

```
$ chathpc ()> /context
Context: Introduction to Kokkos
$ chathpc (Introduction to Kokkos)> What is LayoutLeft?
LayoutLeft refers column-major layout where consecutive entries in
      the same column of a 2-D array are contiguous in memory.
```

Fine-Tuning and Testing

Supervised by HPC experts

- Fine-tuning consists of a JSON or YAML file with a collection of question-context-answer tuples
- Testing data must be different to training (fine-tuning) data.
  Expected answers can be added as part of the testing data for fast testing using ChatHPC

```
[
 {"question": "Can you translate this HIP code to Kokkos?
  __global__ void stencil (int n, float dx, float dy, float *u,
     float *u_new, float *f) {
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   int j = blockIdx.y * blockDim.y + threadIdx.y;
   if(i > 0 && i < n-1 && j > 0 && j < n-1){
    u_new[i][j] = 0.25 * (u[i-1][j] + u[i][j+1]
                  + u[i][j-1] + u[i+1][j] + f[i][j]);
   }
  }",
  "context": "HIP to Kokkos translation",
  "answer":
  "Kokkos::parallel_for(\"stencil\",
     Kokkos::MDRangePolicy<Kokkos::Rank<2>>
            ({1, 1}, {n-1, n-1}),
     KOKKOS_LAMBDA(const int i, const int j) {
      u_new(i,j) = 0.25 * (u(i-1,j) + u(i,j+1)
                   + u(i,j-1 + u(i+1,j) + f(i,j));
  });"
  },
  ...
]
```

Evaluation

Elevating LLM capabilities for HPC targets

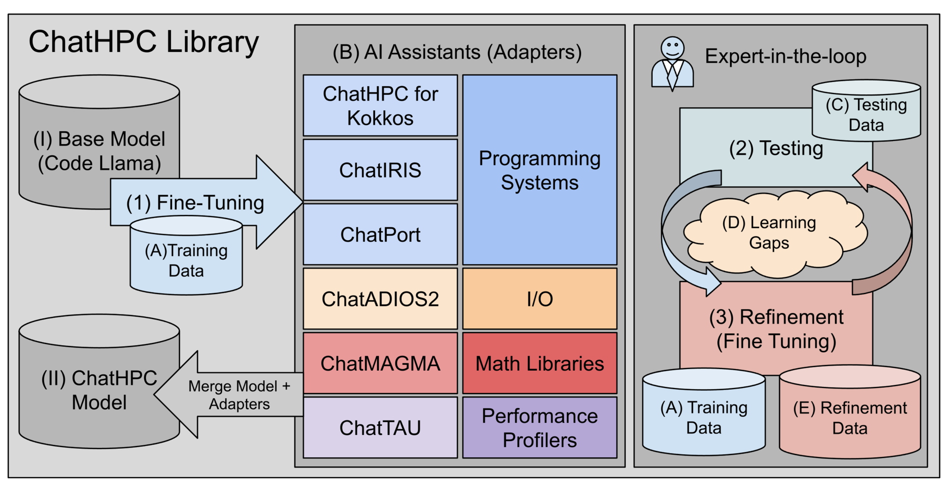
- We measure the performance of the AI assistants as the percentage of the prompts within testing data sets that were answered correctly

ChatHPC AI Assistant	Code Llama	Initial	Refinement	ChatGPT 4o
ChatHPC for Kokkos	7.34%	77.51%	**85.75%**	48.34%
ChatADIOS2	0.00%	40.87%	**100.00%**	25.00%
ChatMAGMA	0.00%	44.26%	**94.93%**	0.00%
ChatIRIS	0.00%	**87.29%**	-	35.00%
ChatPORT	2.54%	20.00%	76.00%	**79.66%**

Impacting correctness with more data

ChatHPC AI Assistant	Extra Training Data used for Refinement	Correctness
ChatHPC for Kokkos	12.33%	17.77%
ChatADIOS2	37.50%	62.85%
ChatMAGMA	51.72%	53.37%
ChatPORT	77.27%	76.68%

ChatHPC Design: How to use it?



3 easy steps

1. **Fine-tuning**
   AI assistant creation
2. **Testing**
   Identify learning gaps
3. **Refinement**
   Filling learning gaps + fine tuning

https://code.ornl.gov/ChatHPC/ChatHPC-project

ChatHPC for Kokkos: An AI Assistant for the C++ Kokkos Portable Front-End

5 contexts/capabilities

1. Documentation
2. Installation
3. Development
4. Parallelization
5. Translation/Portability OpenMP/ACC, CUDA, HIP

Context	Code Llama	ChatHPC for Kokkos (Initial)	ChatHPC for Kokkos (Refinement)	ChatGPT (4o)
Documentation	9.50%	**89.00%**	-	81.00%
Installation	27.20%	**78.00%**	-	45.50%
Development	0.00%	**85.00%**	-	15.14%
Parallelization	0.00%	45.00%	**90.90%**	66.70%
Translation	0.00%	55.58%	**85.85%**	33.40%
OpenACC*	0.00%	53.20%	**87.20%**	41.30%
CUDA*	0.00%	58.70%	**83.50%**	31.20%

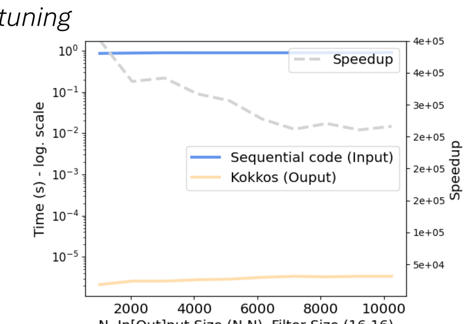
**No OpenACC and CUDA code used for fine-tuning*

Translate the next CUDA code to Kokkos

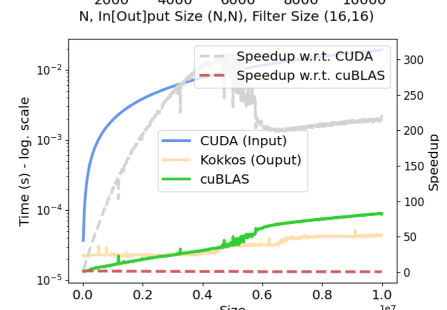
```
__global__ void sdot(int n, float *x, float *y, float *result) {
  int i = threadIdx.x + blockIdx.x * blockDim.x;
  float update = 0.0;
  if (i < SIZE) {
    update = x[i] * y[i];
    atomicAdd(result, update);
  }
}
```

Chat for Kokkos output:

```
Kokkos::parallel_reduce(SIZE,
  KOKKOS_LAMBDA(int i, float &update){
    update += x(m) * y(m);
  },
  Kokkos::Sum<float>(result));
```



$3x10^5$speedup on AI convolution



$3x10^3$ Speedup on DOT product Competetive w. r. t. cuBLAS

Acknowledgments

This research used resources of the Oak Ridge Leadership Computing Facility and the Experimental Computing Laboratory (ExCL) at the Oak Ridge National Laboratory, which is supported by the Office of Science of the US Department of Energy under Contract No. DE-AC05-00OR22725. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR) as part of the Advancements in Artificial Intelligence for Science program, Ellora and Durban projects, the Next Generation of Scientific Software Technologies program, S4PST and PESO projects, and the Scientific Discovery through Advanced Computing (SciDAC) program, RAPIDS2 SciDAC Institute for Computer Science, Data, and Artificial Intelligence.

ChatIRIS: Taskifycation of HPC Applications Using IRIS

- IRIS is a portable task-based runtime for extreme heterogenous systems
- ChatIRIS: Assist developers in the taskifycation of OpenMP, CUDA and HIP applications using IRIS
- No need of refinement

Translate the next CUDA code to IRIS:

```
alpha = 2.0;
X = (float *) malloc(SIZE * sizeof(float));
Y = (float *) malloc(SIZE * sizeof(float));
Z = (float *) malloc(SIZE * sizeof(float));
for (int i = 0; i < SIZE; i++) { X[i] = i; Y[i] = i;}
cudaMalloc((void**)&d_x, SIZE * sizeof(float));
cudaMalloc((void**)&d_y, SIZE * sizeof(float));
cudaMalloc((void**)&d_z, SIZE * sizeof(float));
// Initialization of arrays
cudaMemcpy(d_x, x, SIZE* sizeof(float), cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, SIZE* sizeof(float), cudaMemcpyHostToDevice);
int threads = 512;
int blocks = floor(SIZE/threads) + 1;
saxpy<<<blocks, threads>>>(alpha, d_x, d_y, SIZE);
cudaDeviceSynchronize()
sdot<<<blocks, threads>>>(d_z, d_x, SIZE);
cudaDeviceSynchronize()
cudaMemcpy(z, d_z, SIZE * sizeof(float), cudaMemcpyDeviceToHost);
cudaFree(d_x);
cudaFree(d_y);
cudaFree(d_z);
free(X);
free(Y);
free(Z);
```

ChatIRIS output:

```
// Architecture-agnostic IRIS task code
alpha = 2.0;
iris_init(&argc, &argv, 1);
iris_mem iris_x;
x = (float *) malloc(SIZE * sizeof(float));
iris_data_mem_create(&iris_x, x, SIZE * sizeof(float));
iris_mem iris_y;
y = (float *) malloc(SIZE * sizeof(float));
iris_data_mem_create(&mem_Y, Y, SIZE * sizeof(float));
iris_mem iris_z;
z = (float *) malloc(SIZE * sizeof(float));
iris_data_mem_create(&mem_z, z, SIZE * sizeof(float));
// Initialization of arrays
iris_graph graph;
iris_graph_create(&graph);
iris_task saxpy_task;
iris_task sdot_task;
iris_task_create(&saxpy_task);
iris_task_create(&sdot_task);
void* saxpy_params[3]={&alpha, &iris_x, &iris_y};
int saxpy_params_info[3]={sizeof(A), iris_w, iris_r};
iris_task_kernel(task0, "saxpy", 1, NULL, &SIZE, NULL, 3,
     saxpy_params, saxpy_params_info);
void* sdot_params[2]={&iris_z, &iris_x};
int sdot_params_info[2]={iris_w, iris_r};
iris_task_kernel(task1, "sdot", 1, NULL, &SIZE, NULL, 2,
     sdot_params, sdot_params_info);
iris_graph_task(graph, saxpy_task, iris_default, NULL);
iris_graph_task(graph, sdot_task, iris_default, NULL);
iris_graph_submit(graph,iris_any,1);
free(X);
free(Y);
free(Z);
iris_finalize();
```

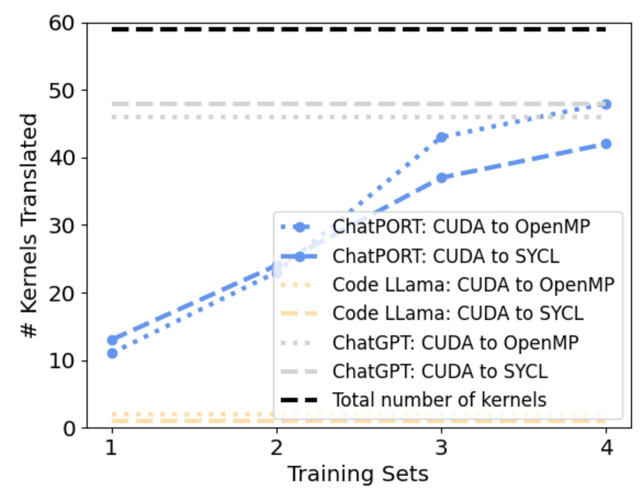
**No HIP code used for fine-tuning*

Context	Code Llama	ChatIRIS	ChatGPT 4o
Documentation	0.00%	**95.00%**	70.00%
OpenMP -> IRIS task code	0.00%	**82.00%**	10.00%
CUDA -> IRIS task code	0.00%	**90.00%**	50.00%
HIP -> IRIS task code*	0.00%	**81.66%**	10.00%

ChatPORT: Porting CUDA to SYCL/OpenMP Offloading for HPC Kernels

- ChatPORT uses benchmarks from HecBench with hundreds of kernels implemented in CUDA, HIP, SYCL and OpenMP Offloading
- ChatPORT provides competitive performance w.r.t. ChatGPT 4o using only 22 kernels for fine-tuning

Training Set	#Kernels	Benchmark (kernel) used
1	5	accuracy, geodesic, lr, maxpool3d, perplexity
2	10	damage, knn, heat2d, laplace3d, md
3	15	advCubatureHex3D, backprop, channelShuffle, lr, meanshift
4	22	chemv, clenergy, pathfinder, pointwise, swish, stddev, tissue



ChatMAGMA: Porting HPC Applications to MAGMA

- **MAGMA** is a portable and heterogeneous **BLAS/LaPACK** library for **Intel/NVIDIA/AMD** accelerators
- **ChatMAGMA** assists application developers in the porting of vendor-specific **BLAS/LaPACK**
- codes (codes using Intel **MKL, NVIDIA cuBLAS** and **cuSolver**, or **AMD hipBLAS** and **hipSolver)** to **MAGMA** library
- High level of trusthworthiness w.r.t. Code Llama (no fine-tuning) and **ChatGPT 4o**
- Fine-tuning time less than 10 min. on 2x **NVIDIA** GPUs
- This work was made in honor of **Stan Tomov**, father of the **MAGMA** library

Translate the next NVIDIA cuSolver code to MAGMA:

```
// NVIDIA cuSolver (GPU-only) dgetrf code
int lwork=0;
double *dwork;
int *dipiv;
double *dinfo;
cusolverDnHandle_t handle;
cusolverDnCreate(&handle);
double *hA=(double *)malloc(n*n*sizeof(double));
cusolverDnDgetrf_bufferSize(handle,n,n,dA,n,&lwork));
cudaMalloc((void **)&dA,n*n*sizeof(double));
cudaMalloc((void **)&dwork, sizeof(double)*lwork);
cudaMalloc((void **)&dipiv, n*sizeof(int));
cudaMalloc((void **)&dinfo, sizeof(int));
cudaMemcpy(dA,hA,n*n*sizeof(double),cudaMemcpyHostToDevice);
cusolverDnDgetrf(handle,n,dA,n,dwork,dipiv,dinfo);
cudaMemcpy(hA,dA,n*n*sizeof(double),cudaMemcpyDeviceToHost);
cudaFree(dA);
cudaFree(dwork);
cudaFree(dipiv);
cudaFree(dinfo);
free(hA);
cusolverDnDestroy(handle);
```

ChatMAGMA output:

```
// MAGMA (CPU+GPU) dgetrf code
magma_queue_t queue;
magma_device_t cdev;
magma_getdevice(&cdev);
magma_queue_create(cdev,&queue );
double *hA;
magmaDouble_ptr dA;
magma_int_t *ipiv;
magma_int_t info;
magma_dmalloc_cpu(&hA,n*n);
magma_dmalloc(&dA,n*n);
magma_imalloc_cpu(&ipiv,n);
magma_dsetmatrix(n,n,hA,n,dA,n,queue);
magma_dgetrf_gpu(n,n,dA,n,ipiv,&info);
magma_dgetmatrix(n,n,dA,n,hA,n,queue);
magma_free(dA);
magma_free_cpu(hA);
magma_free_cpu(ipiv);
magma_queue_destroy(queue);
```

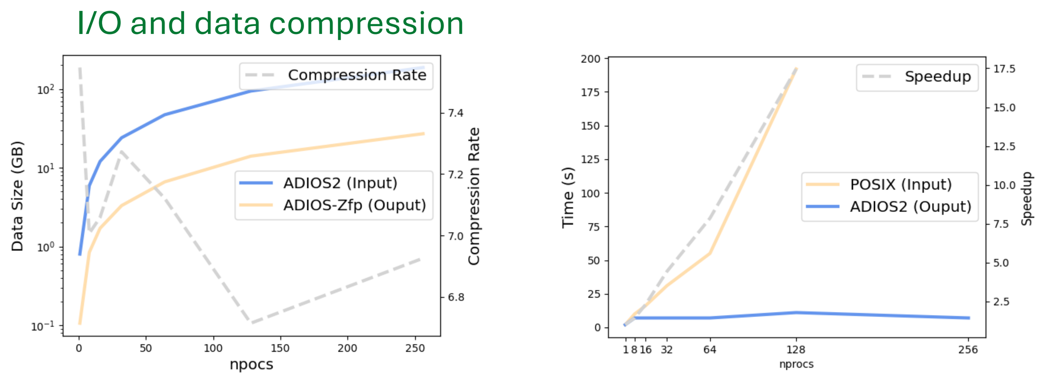
**No AMD hipBLAS or hipSolver code used for fine-tuning*

Context	Code Llama	ChatMAGMA (Initial)	ChatMAGMA (Refinement)	ChatGPT (4o)
Intel MKL	0.00%	37.50%	**93.00%**	0.00%
NVIDIA cuBLAS/cuSolver	0.00%	50.00%	**96.50%**	0.00%
AMD hipBLAS/hipSolver*	0.00%	45.30%	**95.30%**	0.00%

ChatADIOS2: HPC I/O for Applications using ADIOS2

- ADIOS2 is a C++ library to enable HPC writes to parallel file systems and in-memory communication
- ChatADIOS2 assist developers with ADIOS2 documentation, definition of ADIOS2 variables, data compression and parallelization

High scalability and speedup on ORNL's Frontier for parallel I/O and data compression



Context	Code Llama	ChatADIOS2 (Initial)	ChatADIOS2 (Refinement)	ChatGPT (4o)
Introduction	0.00%	77.00%	**100.00%**	33.00%
Variable Definition	0.00%	50.00%	**100.00%**	33.00%
Data Compression	0.00%	25.00%	**100.00%**	0.00%
Parallelization	0.00%	0.00%	**100.00%**	**100.00%**

Future Work: Facing the Challenges

Towards a fully AI-Assisted HPC Ecosystem

- Expand on fine-tuning using larger windows for longer codes
- Agentic AI for a better HPC-AI interoperability
- Elevate trustworthiness levels
- LLM multi-modality for HPC targets

Observations

- Fine-tuning: We demonstrated the effectiveness of finetuning pretrained and relatively small LLMs (7-billion parameter Code Llama model) by using expert-supervised data as a cost-effective approach to rapidly creating trustworthy HPC capabilities with high levels of correctness
- Self-learning: Because of the recurring repetitive patterns found in multiple HPC cases, AI assistants can learn new capabilities, such as portability of AMD math libraries (ChatMAGMA), translation of CUDA or HIP codes (ChatHPC for Kokkos and ChatIRIS), or N-dimensional data definition (ChatADIOS2), without being provided specific training data for those cases, thereby reducing the required size of the training datasets
- Expert-in-the-loop: Performance of AI assistants improves considerably when human expertise is integrated into the fine-tuning process to create the training data and identify learning gaps, although experts may not be always available
- Accessibility: ChatHPC infrastructure is accessible to everyone and requires only modest computational resources (a few minutes on a node with two NVIDIA GPUs). Also, we demonstrated that the data required for users to create new capabilities is relatively small
- Impact on HPC: Apart from elevating the productivity of HPC software by using AI assistants in critical tasks (parallelization, portability, optimization, scalability, and instrumentation), the AI assistants can optimize codes from those passed as input to achieve important speedups (of up to 300×) and scalability improvements (of up to 17.5×) for different domains
- Leveraging HPC efforts: ChatHPC benefits from the important efforts made by the HPC community in the past decade to provide highly productive and portable solutions (e.g., Kokkos, ADIOS2), and the code transformations made by the AI assistants provide the capabilities (scalability, speedup, portability) that modern HPC requires.