OAK RIDGE National Laboratory

SCA 2026 Supercomputing Asia
Gathering the Best of HPC in Asia
HPC Asia 2026

ChatMPI:
LLM-Driven MPI Code Generation for HPC Workloads

SCA/HPCAsia'26, Osaka, Japan
January 26-29, 2026

PRESENTED BY

Pedro Valero-Lara
PhD. Sr Computer Scientist <valerolarap@ornl.gov>

U.S. DEPARTMENT OF ENERGY

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

Introduction

Motivation

AI-driven MPI parallelization that provides good performance through the effective use of HPC fundamentals (e.g., problem decomposition and communication patterns)

Hypothesis

"LLMs can parallelize sequential programs into fully functional and well-performing MPI codes"

Objectives

1. Training LLMs to generate functional MPI codes (i.e., codes that implement MPI syntax correctly and can be compiled and run) from a collection of sequential codes that implement essential HPC workloads
2. Enabling LLMs to implement well-performing MPI codes that leverage HPC fundamentals by making informed decisions regarding work decomposition to minimize communication and maximize performance and scalability when possible

Contribution

ChatMPI, an AI assistant designed to support HPC developers in the MPI parallelization of sequential C codes

ChatHPC

Infrastructure:

- A CLI Python library on top of:
  - Code Llama, PyTorch, and LoRA
- The ChatHPC library makes:
  - Fine-tuning, Testing, Refinement, Merge, and Inference

Ecosystem:

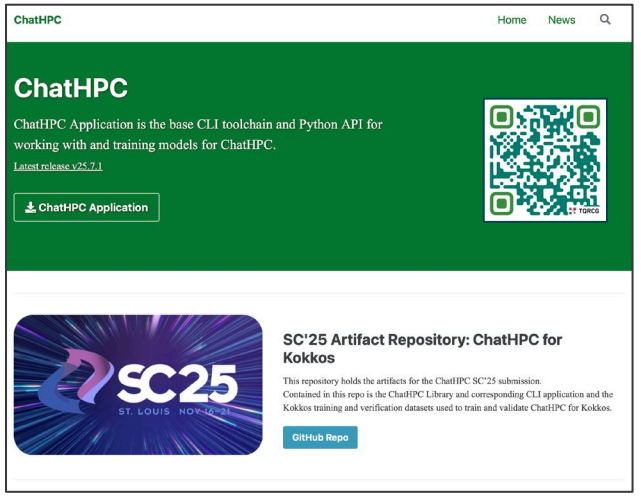
- A collection of AI assistants leveraging existing HPC efforts for multiple domains:
  - Programming Models, Math Libraries, I/O, Tooling, etc.
- And priorities:
  - Parallelization, Portability, Optimization, Evaluation, etc.

ChatHPC CLI:

```
$ chathpc train   # Finetune the assistent.
$ chathpc verify  # Verify the assistent on training set.
$ chathpc test    # Test the asssistent 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.
```



SC25 St. Louis, MO | hpc ignites.

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

Pedro Valero Lara
Oak Ridge National Laboratory (ORNL)
Knoxville, USA
valerolarap@ornl.gov

Aaron Young
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
youngar@ornl.gov

Jeffrey S. Vetter
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
vetter@ornl.gov

Zheming Jin
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
jinz@ornl.gov

Swaroop Pophale
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
pophaless@ornl.gov

Mohammad Alaul Haque Monil
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
monilm@ornl.gov

Keita Teranishi
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
teranishik@ornl.gov

William F. Godoy
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
godoywf@ornl.gov

Abstract

ChatHPC democratizes large language models for the high-performance computing (HPC) community by providing the infrastructure, ecosystem, and knowledge needed to apply modern generative AI technologies to rapidly create specific capabilities for critical HPC components while using relatively modest computational resources. Our divide-and-conquer approach focuses on creating a collection of reliable, highly specialized, and optimized AI assistants for HPC based on the cost-effective and fast Code Llama fine-tuning processes and expert supervision. We target major components of the HPC software stack, including programming models, runtimes, I/O, tooling, and math libraries. Thanks to AI, ChatHPC provides a more productive HPC ecosystem by boosting important tasks related to portability, parallelization, optimization, scalability, and instrumentation, among others. With relatively small datasets (on the order of KB), the AI assistants, which are created in a few minutes by using one node with two NVIDIA H100 GPUs and the ChatHPC library, can create new capabilities with Meta's 7-billion parameter Code Llama base model to produce high-quality software with a level of trustworthiness of up to 90% higher than the 1.8-trillion parameter OpenAI ChatGPT-4o model for critical programming tasks in the HPC software stack.

CCS Concepts

• **Computing methodologies** → **Natural language processing**.

Keywords

Large Language Models, Productivity, Trustworthiness, High Performance Computing.

ACM Reference Format:

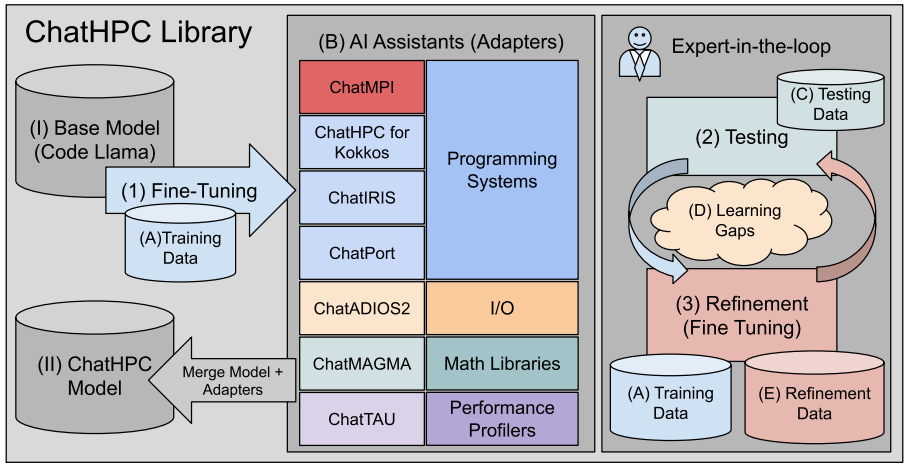
Pedro Valero Lara, Aaron Young, Jeffrey S. Vetter, Zheming Jin, Swaroop Pophale, Mohammad Alaul Haque Monil, Keita Teranishi, and William F. Godoy. 2025. ChatHPC: Building the Foundations for a Productive and Trustworthy AI-Assisted HPC Ecosystem. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3712285.3759787

1 Introduction

High-performance computing (HPC) applications demand effective parallelization, optimization, and portability to leverage modern heterogeneous architectures. However, these tasks often require significant expertise and labor, which pose challenges for developers when rapid turnaround is required. Recent advances in large language models (LLMs) offer a promising solution by automating and enhancing complex programming tasks [1–4], thereby accelerating the development process.

In this paper, we introduce ChatHPC: a process and tool chain for developing AI assistants fine-tuned from Code Llama [1] and specifically designed to boost productivity in HPC software development. Our initial ChatHPC models target three critical areas: parallelizing sequential code, optimizing existing parallel implementations, and

ChatMPI: Design & Implementation



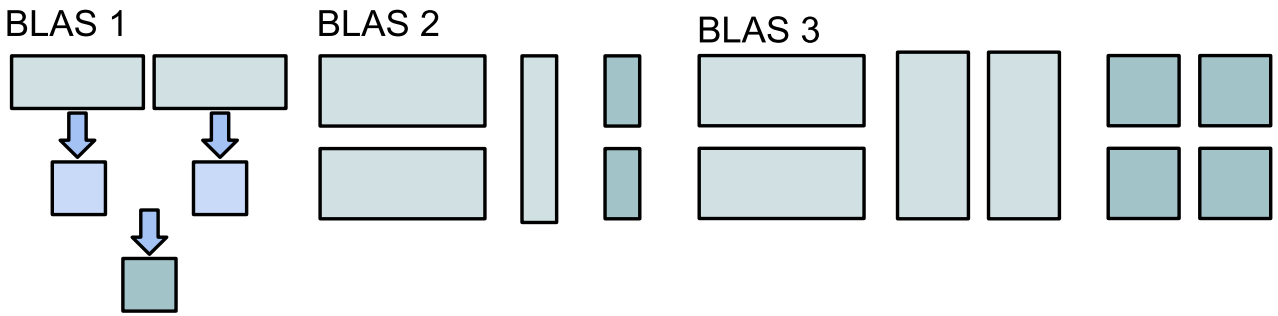
A simple 3-step iterative process:

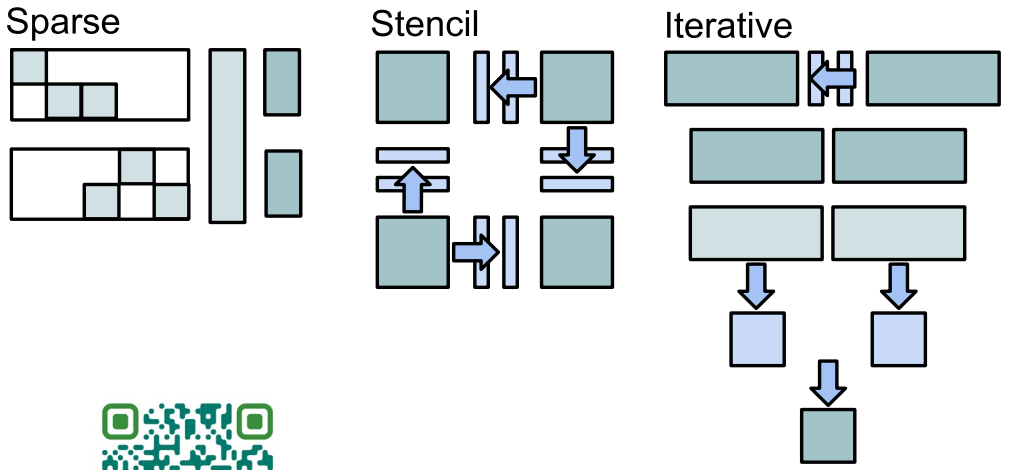
1. **Fine-tuning -> AI assistant (optimizers)**
   - Training Data
   - The better the data the better the model
   - It is more about quality than quantity
2. **Testing -> Learning gaps**
   - Testing Data != Training Data
   - Testing Data may contain question about non-fined-tuned capabilities
3. **Refinement -> More capable AI assistants**
   - Expert-Supervised Refinement Data

AI assistants can be merge into a single LLM: ChatHPC model

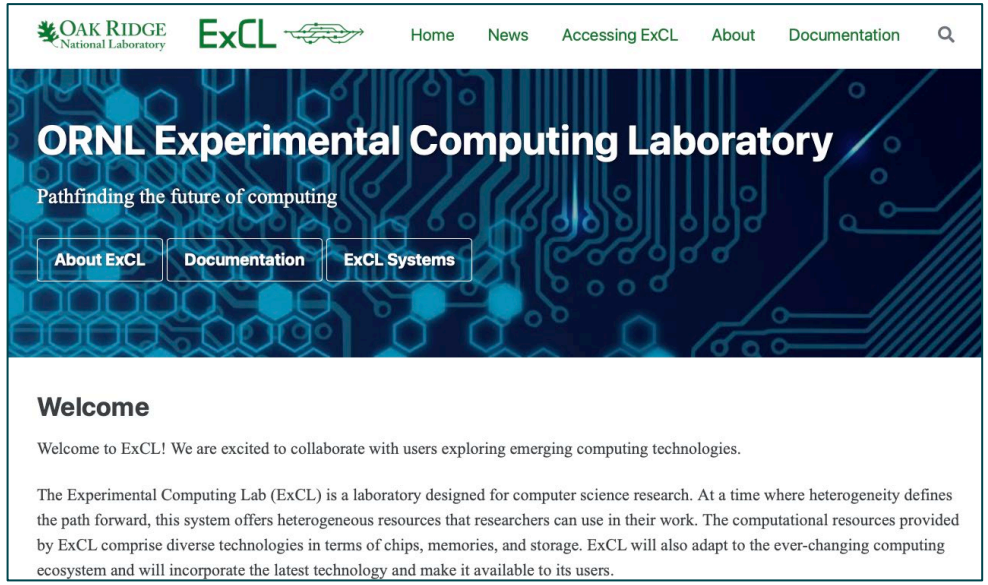
```
[
 {"question": "Can you parallelize this sequential code to MPI?
  int n = 500;
  float *u = malloc(n*n*sizeof(float));
  float *u_new = malloc(n*n*sizeof(float));
  for (int i = 1; i < n-1; i++) {
    for (int j = 1; j < n-1; j++) {
      u_new[i*n+j] = 0.25*(u[(i-1)*n+j]+ ...
    }
  }
  free(u);
  free(u_new);
  "context": "Sequential to MPI translation",
  "answer": "
  int size, rank;
  MPI_Status status;
  MPI_Init(NULL, NULL);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  int n = 500;
  int local_n = ceil(500/size) + 2;
  float *u = malloc(n*local_n*sizeof(float));
  float *u_new = malloc(n*local_n*sizeof(float));
  if (rank > 0) {
    MPI_Send(&u[1],1,MPI_FLOAT,rank-1,0,MPI_COMM_WORLD);
    MPI_Recv(&u[0],1,MPI_FLOAT,rank-1,0,MPI_COMM_WORLD,&status);
  }
  if (rank < size-1) {
    MPI_Recv(&u[n-1],1,MPI_FLOAT,rank+1,0,MPI_COMM_WORLD, &status);
    MPI_Send(&u[n-2],1,MPI_FLOAT,rank+1,0,MPI_COMM_WORLD);
  }
  for (int i = 1; i < local_n-1; i++) {
    for (int j = 1; j < n-1; j++) {
      u_new[i*n+j] = 0.25*(u[(i-1)*n+j] + ...
    }
  }
  free(u);
  free(u_new);
  MPI_Finalize();
  },
]
```

Analysis



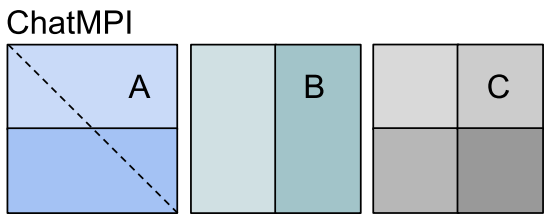


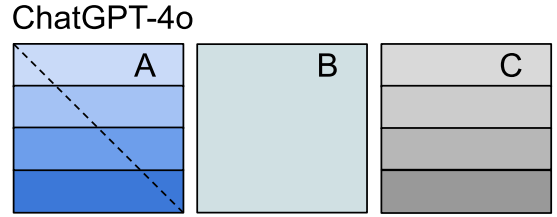
- Intel Xeon Gold 6130 CPU (Skylake) 32-core
- AI assistants (~100MB)
- Training/Testing data (~KB)
- Fine-tuning (Code Llama [7B]):
  - No more than 15 min. in a one NVIDIA 2x H100 node
- Evaluation
  - Execution time and scalability
  - Evaluation data is different to training data
- Compare against
  - 7 billion parameter Meta's Code Llama base model (no fine-tuning)
  - 1.8 trillion parameter OpenAI's ChatGPT-4o model
  - ChatGPT-5 Reasoning LLM

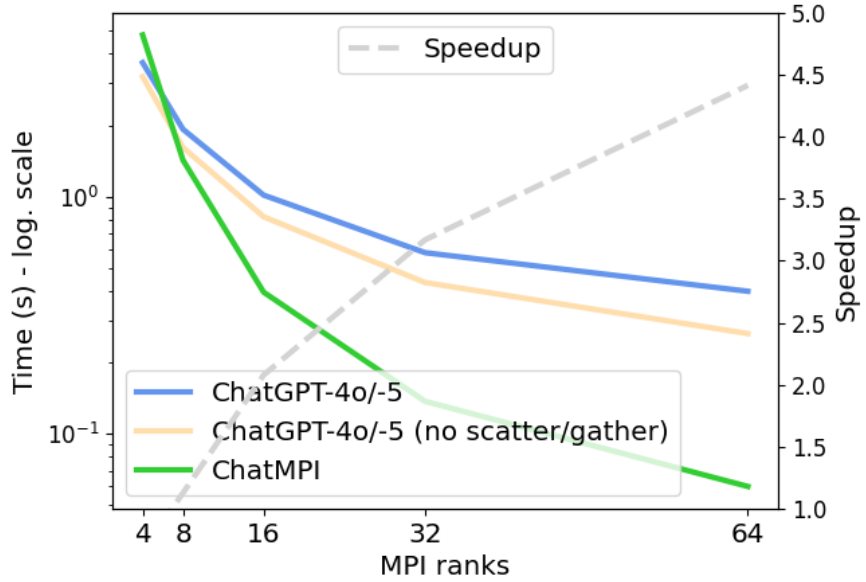


BLAS

- **Training**
  - Level 1: DOT and AXPY
  - Level 2: GEMV and TRMV
  - Level 3: GEMM and TRMM
  - Balanced vector/matrix decomposition
  - No unnecessary communication
    - Initial scatter and final gather
- **Testing**
  - Rest of BLAS
- **Data**
  - Training (~5%) vs Testing (~95%)
- **Results (SYMM)**
  - ChatGPT-4o and -5 use the same matrix decomposition
    - Matrix B is replicated across ranks
  - ChatMPI uses an even computation/data distribution
    - Speedup w.r.t. ChatGPT codes
    - Scalability increases when adding more ranks







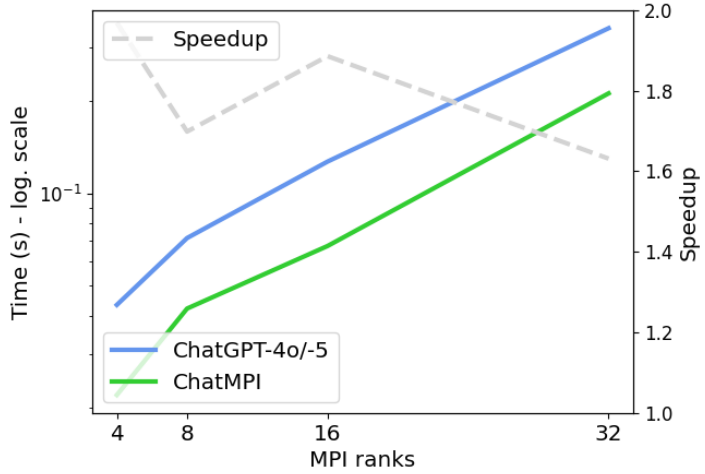
Sparse Linear Algebra (SpMV)

- **Data**
  - Each case (format) must be trained
  - Important differences in the implementations using different formats
- **Results (CSR)**
  - All models use a similar strategy
    - Replicate the matrix across MPI processes (communication avoiding)
  - ChatGPT-4o and -5
    - Initialize vectors in rank 0 and distribute across all ranks
  - ChatMPI
    - Initializes vectors across all the ranks

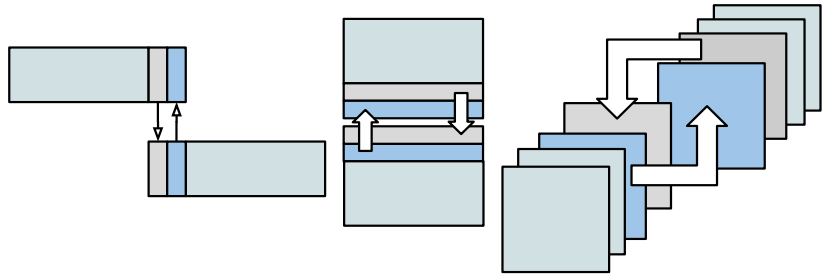
```c
int n, nnz;
MPI_Init(NULL, NULL);
int rank, size;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
int local_n = n / size;
int start = rank * local_n;
int end = start + local_n;
float * x = malloc(n * sizeof(float));
float *y = malloc(local_n * sizeof(float));
float *val = malloc(nnz * sizeof(float));
int *col = malloc(nnz * sizeof(int));
int *row = malloc((n + 1) * sizeof(int));
// Initialization of the matrix and vectors
for (int i = start; i < end; i++) {
 float sum = 0.0f;
 for (int j = row[i]; j < row[i + 1]; j++) {
  sum += val[j] * x[(int)col[j]];
 }
 y[i - start] = sum;
}
free(x);
free(y);
free(val);
free(col);
free(row);
MPI_Finalize();
```

```c
int nrow, ncol, nnz;
MPI_Init(&argc, &argv);
int rank, size;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
int local_ncol = ncol / size;
int start_col = rank * cols_per_proc;
int end_col = start_col + local_ncol;
float *x = malloc(ncol * sizeof(float));
float *y_local = calloc(nrow, sizeof(float));
int *col_ptr = col_ptr = malloc((ncol + 1) * sizeof(int));
int *row_ind = malloc(nnz * sizeof(int));
float *val = malloc(nnz * sizeof(float));
// Initialization of the matrix and vectors
for (int i = start_col; i < end_col; i++) {
 for (int j = col_ptr[i]; j < col_ptr[i + 1]; j++) {
  y_local[row_ind[j]] += val[j] * x[i];
 }
}
float *y_global = NULL;
if (rank == 0) y_global = malloc(nrow * sizeof(float));
MPI_Reduce(y_local, y_global, nrow, MPI_FLOAT, MPI_SUM, 0,
↪  MPI_COMM_WORLD);
free(x);
free(y_local);
free(row_ind);
free(col_ptr);
free(val);
MPI_Finalize();
```

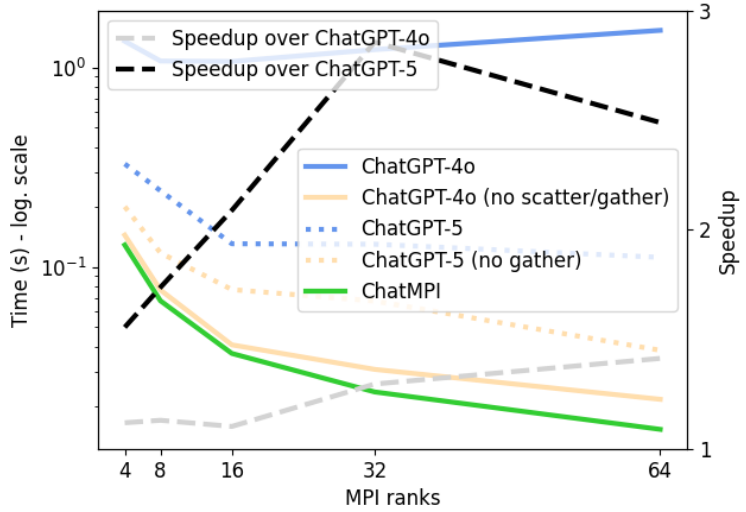
Fig. 7. MPI pseudocode for SpMV using (left) CSR and (right) CSC formats.



Stencil Solvers (Euler)



- **Training (simplest cases)**
  - 1D 3-point, 2D 5-point, and 3D 9-point
- **Testing (more complex)**
  - Triagonal Matrix-Vector multiplication
  - 2D 9-point
  - 3D 11-point
- **Data**
  - Training (~50%) vs Testing (~50%)



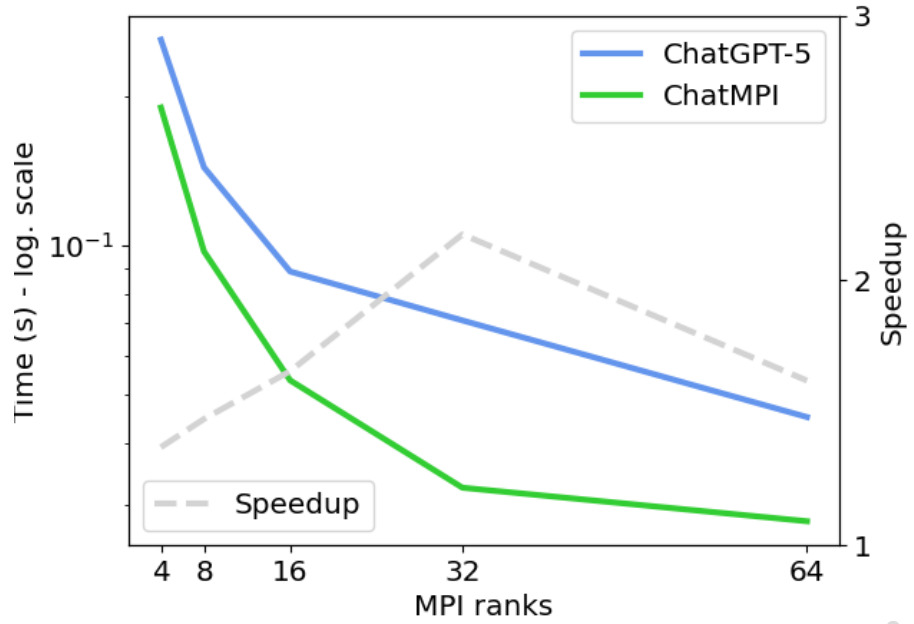
- **Results**
  - ChatGPT 4-o
    - Only generates functional code for the 1st test case using non-blocking communication
  - ChatGPT-5
    - Generates functional code for all test cases using blocking (MPI_Sendrecv) communication for the 1st test case and non-blocking for the rest
  - ChatMPI:
    - Generates MPI codes using blocking communication

Iterative Solvers (CG)

- **Training (simplest cases)**
  - No training data needed
- **Results**
  - ChatGPT 4-o
    - Does not generate any functional code
  - ChatGPT-5
    - Uses non-blocking communication and strong syncs
    - Adds an important overhead
  - ChatMPI:
    - Generates MPI codes using techniques trained previously
    - Communication avoiding
    - Blocking communication
    - …

```
int size, rank;
MPI_Init(NULL, NULL);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
int n = ceil(5000/size) + 2;
float alpha, local_sum, global_sum;
float *u = malloc(n*sizeof(float));
float *u_new = malloc(n*sizeof(float));
float *a1 = malloc(n*sizeof(float));
float *a2 = malloc(n*sizeof(float));
float *a3 = malloc(n*sizeof(float));
// Initialization of matrix and vectors
// MPI point-to-point communication
if (rank > 0) {
  MPI_Send(&u[1], 1, MPI_FLOAT, rank - 1, 0, MPI_COMM_WORLD);
  MPI_Recv(&u[0], 1, MPI_FLOAT, rank - 1, 0, MPI_COMM_WORLD, &status);
}
if (rank < size-1) {
  MPI_Recv(&u[n-1], 1, MPI_FLOAT, rank + 1, 0, MPI_COMM_WORLD, &status);
  MPI_Send(&u[n-2], 1, MPI_FLOAT, rank + 1, 0, MPI_COMM_WORLD);
}
// Tridiagonal matrix-vector multiplication
for (int i = 1; i < n-1; i++) {
  u_new[i] = a1[i]*u[i+1] + a2[i]*u[i]  + a3[i]*u[i-1];
}
// BLAS 1 AXPY
for (int i = 0, i < n, i++) {
  u_new[i] = alpha * u[i] + u_new[i];
}
//BLAS 1 DOT
for (i = 0, i < n, i++) {
  local_sum += u[i] * u_new[i];
}
// MPI collective communication
MPI_Reduce(&local_sum, &global_sum, 1, MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD);
free(u); ...
MPI_Finalize();
```

Fig. 11. MPI pseudocode for the main steps of the CG algorithms.



Remarks

ChatMPI		Correctness				Peak ChatMPI Speedup over	
HPC Domain	Training Size	Code Llama	ChatMPI	ChatGPT-4o	ChatGPT-5	ChatGPT-4o	ChatGPT-5
BLAS Level 1, 2, 3	Small (5%)	0%	100%	100%	100%	4x	4x
Sparse (SpMV)	Large (100%)	0%	100%	100%	100%	1.9x	1.9x
1D, 2D, 3D Stencil (Euler)	Medium (50%)	0%	100%	33%	100%	1.4x	2.8x
Iterative (CG)	No data needed	0%	100%	0%	100%	-	2.1x

- **Training**
  - Very dependent on the target
- **Correctness**
  - ChatMPI and ChatGPT-5 provide high levels of trustworthiness
  - Code Llama is not exposed to MPI programming and ChatGPT-4o is partially exposed
- **Performance**
  - Up to 4x better performance and scalability
  - Workload decomposition, communication avoiding, blocking communication, …

Future Work

3 Pillars

- **Multimodality**
  - Simplify the interaction with AI for HPC domains
- **Reasoning**
  - Generate highly optimized HPC codes across domains and technologies
- **Agentic**
  - Connect HPC and AI for complex problems

Observations

Fine-tuning

- Very effective, if you have the data

MPI Performance

- Training on fundamentals
  (BTW: much better than ChatGPT)

Overfitting

- Is that a problem for domain-specific AI assistants?

Self-learning

- Train on simple problems and let LLMs to deal with more complex problems
- Take advantage from similarities (similar patterns)

Expert-in-the-loop

- Recommendable for domain-specific LLMs

Accessibility

- One NVIDIA GPU (better if you have 2), data (~KB), and 15 minutes

Simplify prompting

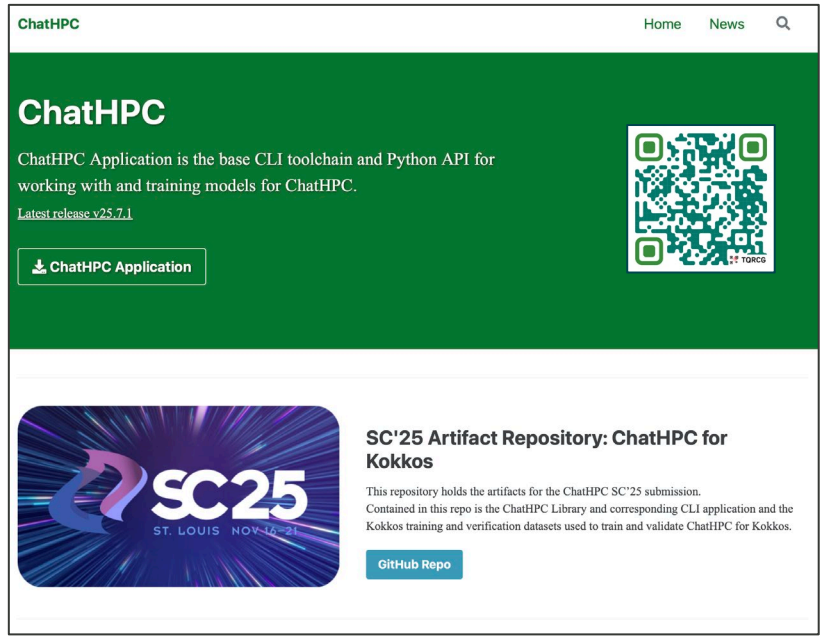
- No prompt engineering, more productive, non-experts can generate expert level responses

Data

- Very demanding depending on the target (SpMV) or not (Stencil, BLAS, Iterative)

Resources, Community, and Announcements

- **ChatHPC project:** https://ornl.github.io/ChatHPC/
- **ChatHPC meetings:**
  - We organize a community meeting once a month. Join us!
- **Workshops:**
  - LLM4HPC (co-located with ISC-HPC'26): https://ornl.github.io/events/llm4hpc2026/
  - LLM4HPCAsia (collocated with SCA/HPC Asia): https://ornl.github.io/events/llm4hpcasia2026/
- **Special Issue at International Journal of Parallel Programming**
  - High-Productivity Programming Systems for HPC Applications: https://link.springer.com/collections/gjbahhgce



Use ChatHPC!

OAK RIDGE National Laboratory

SCA 2026 Supercomputing Asia — Gathering the Best of HPC in Asia

HPC Asia 2026

ChatMPI:
LLM-Driven MPI Code Generation for HPC Workloads

Oh-Kini!

Thanks!

PRESENTED BY

Pedro Valero-Lara

PhD. Sr Computer Scientist <valerolarap@ornl.gov>

U.S. DEPARTMENT OF ENERGY

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

Acknowledgments:

This research used resources of the Oak Ridge Leadership Computing Facility and the Experimental Computing Laboratory 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 US Department of Energy, Office of Science's Advanced Scientific Computing Research program 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.