VisionHPC:
AI Generation of HPC Codes from High-Level Mathematical Abstractions

OAK RIDGE National Laboratory | SC26 CHICAGO NOV 15–20 | Genesis Mission | U.S. DEPARTMENT of ENERGY

Motivation

Elevating the abstraction level to mathematics

- **Bridge mathematical formulations (domain scientists) and optimized HPC implementations**, enabling domain scientists to translate scientific concepts directly into high-performance software.
- **Advance an integrated AI-assisted HPC ecosystem** that delivers unprecedented productivity in the development, optimization, and maintenance of HPC applications.
- **Leverage decades of HPC software expertise** by building upon mature numerical libraries, programming models, and optimization techniques to generate robust, portable, and high-performance implementations.

A Reliable and Efficient Ecosystem

A robust, open-source, and lightweight framework

- **Built on an efficient open-source AI stack**: Llama 3.2 Vision (11B parameters), SFTTrainer, Unsloth, LoRA, and PyTorch.
- **Low computational requirements**: Llama 3.2 Vision (13 GB), AI assistant (hundreds of MB), training/testing datasets (hundreds of MB), and image sizes on the order of hundreds of KB.
- **Rapid AI assistant creation**: Fine-tuning and deployment can be completed in **less than 15 minutes** on a single **NVIDIA H100** GPU.

Extending ChatHPC Capabilities to Multimodality

Accelerating production of AI assistants for HPC

- **An easy-to-use Python front end** for creating, fine-tuning, and evaluating domain-specific AI assistants, with a strong emphasis on high-quality training data.

ChatHPC CLI:

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

- **Fine-tuning and evaluation are supervised by HPC experts** guide the fine-tuning and evaluation loop to ensure correctness, robustness, and domain relevance.
- **A simple three-step iterative workflow**:
  1. **Fine-tuning**: Create a specialized AI assistant.
  2. **Testing**: Identify learning gaps and failure cases.
  3. **Refinement**: Address the identified gaps through data curation and additional fine-tuning.

Compared with Frontier LLMs

Superior results faster, at lower cost, and with modest resources

Framework	LAPACK Generation	LAPACK Generation	MAGMA Code Generation	SLATE Code Generation
VisionHPC	**98%**	**92.5%**	**89%**	**90.5%**
Code Llama	Text-only	0%	0%	0%
ChatGPT	66%	0%	0%	0%
Claude	0%	0%	0%	0%
OSS	Text-only	0%	0%	0%
MathGPT	40%	0%	0%	0%

- Although some frontier multimodal models, such as ChatGPT and MathGPT, have been exposed to matrix equations and can generate correct LAPACK specifications in some cases, all evaluated models struggle to generate HPC code targeting LAPACK libraries. ChatGPT and Claude achieve the best overall performance among the baseline models, but neither is able to produce fully functional implementations.

Redefining HPC Code Development in the AI Era

Position math as a front end for HPC code generation

- **Four essential domains tested:**

Linear Algebra (Level 1, 2, and 3 BLAS)

$$r = \sum_{i=0}^{N-1} x_i \times y_i$$

$$\forall i \in [0, M-1]: \quad y_i = \beta \times y_i$$
$$y_i = \sum_{j=0}^{i} \alpha \times a_{i,j} \times x_j$$

$$\forall i \in [0, M-1]: \quad \forall j \in [0, N-1]: \quad c_{i,j} = \beta \times c_{i,j}$$
$$c_{i,j} = \sum_{m=0}^{K-1} \alpha \times a_{i,m} \times b_{m,j}$$

PDE (Stencil-based)

$$\forall i \in [1, M-2], \quad \forall j \in [1, N-2], \quad \forall l \in [1, K-2]:$$
$$u_{i,j,l} = c \times (v_{i-1,j,l} + v_{i+1,j,l}) + c \times (v_{i,j-1,l} + v_{i,j+1,l}) + c \times (v_{i,j,l-1} + v_{i,j,l+1}) - v_{i,j,l}$$

CFD (Lattice Boltzmann)

$$\rho = \sum f_i(x,t), \quad \rho u = \sum e_i f_i(x,t),$$

$$f_i^*(x, t+\Delta t) = f_i(x,t) - \Omega \times (f_i(x,t) - f_i^{eq}(x,t)),$$

$$f_i(x + e_i \Delta_t, t + \Delta t) = f_i^*(x, t + \Delta t),$$

Matrix Equations (LAPACK)

Multimodal Input		Output
Image	Text	Text
Matrix Equation	Shape & Storage	LAPACK Specification
$A = LU$, $Ly = b$, $Ux = y$	-	DGESV (int n, int nrhs, double *a, int lda, int *ipiv double *b, int ldb, int info)
$A = LU$, $Ly = b$, $Ux = y$	Band	DGBSV (int n, int kl, int ku, int nrhs, double *ab, int ldab, int *ipiv, double *b, int ldb, int info)
$A = LU$, $Ly = b$, $Ux = y$	Tridiagonal	DGTSV (int n, int nrhs, double *dl, double *d, double *du, double *b, int ldb, int info)
$A = LL^T$, $Ly = b$, $L^Tx = y$	-	DPOSV('L', int n, int nrhs, double *a, int lda, double *b, int ldb, int info)
$A = LL^T$, $Ly = b$, $L^Tx = y$	Packed	DPPSV('L', int n, int nrhs, double *ap, double *b, int ldb, int info)
$A = LL^T$, $Ly = b$, $L^Tx = y$	Band	DPBSV('L', int n, int nrhs, double *ab, int ldab, double *b, int ldb, int info)
$A = LL^T$, $Ly = b$, $L^Tx = y$	Tridiagonal	DPTSV(int n, int nrhs, double *d, double *e, double *b, double *b, int ldb, int info)

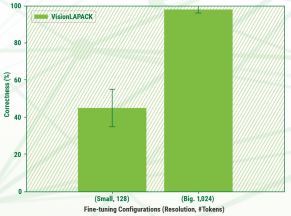
Building an HPC–AI Integrated Environment

Target: reach a 90% correctness level

- **Incremental data selection**: We adopted an incremental fine-tuning strategy to identify the most informative images and source code examples; this approach maximizes code generation quality and minimizes the required training data.
- **Efficient token utilization**: Token requirements scale with problem complexity, ranging from **128 tokens** for linear algebra tasks to **1,024 tokens** for LAPACK code generation.
- **After comprehensive initial training on OpenMP and linear algebra**, the additional data requirements are significantly reduced.

Domain	OpenMP	OpenACC	CUDA/HIP
Linear Algebra	82%	36%	36%
PDE	20%	40%	20%
CFD	25%	12%	12%

- **Image size and token count significantly impact the correctness of generated AI assistants**.
- **LAPACK specification generation requires larger inputs**: images of approximately 1,000 × 1,000 pixels (100–300 KB each) and up to 1,024 tokens during inference to capture the complexity of the mathematical representations.



Join VisionHPC: Contact Us!

Valero-Lara, Pedro: valerolarap@ornl.gov
Aaron Young, younger@ornl.gov
Mohammad Alaul Haque Monil monilm@ornl.gov
William F. Godoy, godoywf@ornl.gov
Keita Teranishi, teranishik@ornl.gov
Jeffrey S. Vetter, vetter@ornl.gov

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 US Department of Energy, Office of Science, 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.

Conclusions

More than a code generator tool

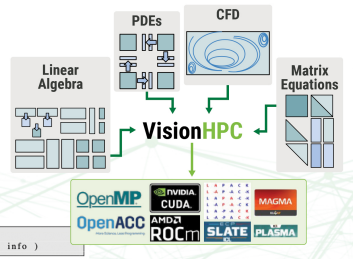
VisionHPC is a new AI-assisted software engineering framework that combines mathematical reasoning, decades of HPC software expertise, and modern agentic AI to bridge the gap between scientific formulations and production-quality HPC implementations while remaining practical, efficient, and broadly accessible.

VisionHPC

AI-Augmented HPC Software Foundations

Scientific reasoning on math foundations for HPC code generation

- **Targeting HPC systems**
  1. **Multi-core CPUs:** OpenMP, PLASMA
  2. **GPUs:** OpenACC, CUDA,HIP, MAGMA
  3. **Distributed memory:** SLATE




```
// ----LAPACK specification
DPOTRF( 'U', int n, double *a, int lda, int info )
```

```
// ----PLASMA implementation
...
plasma_set( PlasmaTuning, PlasmaDisabled );
plasma_set( PlasmaNb, 256 );
double *a = (double*)
   malloc ( (size_t) lda*n*sizeof(double) );
info = plasma_dpotrf( 'U', n, a, lda );
...
```

```
// ----MAGMA implementation
...
magma_dmalloc_pinned( &aux, n*lda );
magma_dmalloc( &a, n*lda );
magma_queue_t queues[2] = { NULL, NULL };
magma_device_t cdev;
magma_getdevice( &cdev );
magma_queue_create( cdev, &queues[0] );
magma_queue_create( cdev, &queues[1] );
magma_dpotrf_expert('U', n, aux, lda,
                    n, lda, &info, queues );
magma_queue_destroy( queues[0] );
magma_queue_destroy( queues[1] );
magma_free_pinned(aux);
...
```

```
// ----SLATE implementation
...
int mpi_size = 0;
int mpi_rank = 0;
int grid_p = 0;
int grid_q = 0;
int provided = 0;
int64_t nb = 256;
slate_mpi_call(
   MPI_Init_thread( &argc, &argv,
   MPI_THREAD_MULTIPLE, &provided ) );
slate_mpi_call(
   MPI_Comm_size( MPI_COMM_WORLD, &mpi_size ) );
slate_mpi_call(
   MPI_Comm_rank( MPI_COMM_WORLD, &mpi_rank ) );
grid_size( mpi_size, &grid_p, &grid_q );
slate::SymmetricMatrix<double> a( slate::Uplo::Upper,
   n, nb, grid_p, grid_q, MPI_COMM_WORLD );
a.insertLocalTiles();
info = slate::potrf( a );
slate_mpi_call( MPI_Finalize() );
...
```

$$r = \sum_{i=0}^{N-1} \| x_i \|_2$$

```
float nrm2(int N, float *x) {
  float result = 0.0;
  #pragma omp parallel for reduction(+:result)
                default(none) private(i) shared(N, x)
  for (int i = 0; i < N; ++i) {
    result += x[i]*x[i];
  }
  return sqrtf(result);
}
```

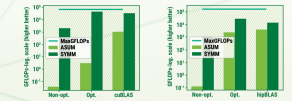
```
float nrm2(int N, float *x) {
  float result = 0.0;
  #pragma acc parallel reduction(+:result)
              present_or_copyin(x[0:N])
  for (int i = 0; i < N; ++i) {
    result += x[i]*x[i];
  }
  return sqrtf(result);
}
```

```
__global__
void nrm2(int N, float *x, float *result) {
  float update = 0.0;
  int i = threadIdx.x+blockIdx.x*blockDim.x
  if (i < N) {
    update = (x[i]*x[i]);
    atomicAdd(result, update);
  }
}
```

Achieving Top Performance

As part of an agent that integrates the capabilities of other LLMs

- **Reasoning and code-centric LLMs:** OSS, GPT Codex, Code Llama
- **Prompting, in-context learning, and fine-tuning**
- **Test bed**: NVIDIA H100 and AMD MI300 GPUs
- **Level 1 BLAS ASUM and Level 3 SYMM**: achieving 1.35× and 2×better performance than cuBLAS and hipBLAS for SYMM computations



- **Top-performing LBM kernels:** achieving up to 80% of peak performance

