HPC-Coder: Modeling Parallel Programs using Large Language Models

Daniel Nichols*, Aniruddha Marathe†, Harshitha Menon†, Todd Gamblin†, Abhinav Bhavele*

* University of Maryland
† Lawrence Livermore National Laboratory
Timeline and Motivation
Timeline and Motivation

August 2021
OpenAI Codex Release
Timeline and Motivation

August 2021
OpenAI Codex Release

November 2022
ChatGPT Release
Timeline and Motivation

August 2021
OpenAI Codex Release

November 2022
ChatGPT Release

I wonder if these are any good at HPC?
Timeline and Motivation

August 2021
OpenAI Codex Release

I wonder if these are any good at HPC?

November 2022
ChatGPT Release

June 2023
HPC-Coder First Release
Timeline and Motivation

August 2021
OpenAI Codex Release

November 2022
ChatGPT Release

August 2023 - Now
CompCoder, HPC-GPT, OMP-GPT,
ParEval, RLPF, Tokompiler

June 2023
HPC-Coder First Release

I wonder if these are any good at HPC?
August 2021
OpenAI Codex Release

November 2022
ChatGPT Release

August 2023 - Now
CompCoder, HPC-GPT, OMP-GPT, ParEval, RLPF, Tokompiler

I wonder if these are any good at HPC?

June 2023
HPC-Coder First Release

and hopefully many more…
Code LLMs are Bad at Parallel Code

- PolyCoder
  - State-of-the-art, open-source, code LLM released in 2022
  - 2.7B parameters and outperformed OpenAI’s Codex
Code LLMs are Bad at Parallel Code

- PolyCoder
  - State-of-the-art, open-source, code LLM released in 2022
  - 2.7B parameters and outperformed OpenAI's Codex

We prompted the LLM to write simple parallel code…

```c
/*
Use MPI to compute the average of X across ranks
and return the result on all ranks.
Use the default communicator.
*/
double mpiAverage(double X) {
```
Code LLMs are Bad at Parallel Code

- **PolyCoder**
  - State-of-the-art, open-source, code LLM released in 2022
  - 2.7B parameters and outperformed OpenAI’s Codex

We prompted the LLM to write simple parallel code…

```c
/*
Use MPI to compute the average of X across ranks and return the result on all ranks.
Use the default communicator.
*/

double mpiAverage(double X) {
    int rank;
    int npprocs;
    double average;
    MPI_Status status;
    MPI_Request request;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &npprocs);

    if (rank == 0) {
        MPI_Recv(&status, 1, MPI_INT, 0, 0,
                  MPI_COMM_WORLD,
                  &request);
        MPI_Send(&X, 1, MPI_INT, 0, 0,
                 MPI_COMM_WORLD);
        MPI_Send(&average, 1, MPI_DOUBLE, 0, 0,
                 MPI_COMM_WORLD);
    }
    return average;
}
```
Code LLMs are Bad at Parallel Code

- PolyCoder
  - State-of-the-art, open-source, code LLM released in 2022
  - 2.7B parameters and outperformed OpenAI's Codex

We prompted the LLM to write simple parallel code...

```c
int rank;
int nprocs;
double average;
MPI_Status status;
MPI_Request request;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

if (rank == 0) {
    MPI_Recv(&status, 1, MPI_INT, 0, 0,
             MPI_COMM_WORLD,
             &request);
    MPI_Send(&X, 1, MPI_INT, 0, 0,
             MPI_COMM_WORLD);
    MPI_Send(&average, 1, MPI_DOUBLE, 0, 0,
             MPI_COMM_WORLD);
}
return average;
```

and usually got incorrect output.

We observed an ability to generate parallel code constructs, but not use them correctly.
Improving LLMs for Parallel Code
RQ 1 – How can we train LLMs to better understand and generate parallel and HPC code?
Improving LLMs for Parallel Code

**RQ 1** – How can we train LLMs to better understand and generate parallel and HPC code?

**RQ 2** – How can we effectively measure the capabilities of LLMs at modelling parallel and HPC code?
Overview of Our Approach

HPC Source Data Collection

Fine-Tuning
- GPT-2
- GPT-Neo
- PolyCoder

Model Selection

Downstream Tasks
- Text Generation
- Code Completion
- OpenMP Labelling
- Performance Prediction
Overview of Our Approach

Collect HPC code data for fine-tuning
HPC Source Data Collection

Fine-Tuning

- GPT-2
- GPT-Neo
- PolyCoder

Model Selection

Downstream Tasks

- Text Generation
- Code Completion
- OpenMP Labelling
- Performance Prediction

RQ 1 – How can we train LLMs to better understand and generate parallel and HPC code?
Overview of Our Approach

**Fine-Tuning**
- GPT-2
- GPT-Neo
- PolyCoder

**Model Selection**

**Downstream Tasks**
- Text Generation
- Code Completion
- OpenMP Labelling
- Performance Prediction

**RQ 1** – How can we train LLMs to better understand and generate parallel and HPC code?
Overview of Our Approach

HPC Source Data Collection

Fine-Tuning
- GPT-2
- GPT-Neo
- PolyCoder

Model Selection

Pick the best model for further testing

Downstream Tasks
- Text Generation
- Code Completion
- OpenMP Labelling
- Performance Prediction

RQ 1 – How can we train LLMs to better understand and generate parallel and HPC code?
**Overview of Our Approach**

- **HPC Source Data Collection**
- **Fine-Tuning**
  - GPT-2
  - GPT-Neo
  - PolyCoder
- **Model Selection**
  - Evaluate on three distinct HPC tasks

**Downstream Tasks**
- Text Generation
  - Code Completion
  - OpenMP Labelling
- Performance Prediction

**RQ 2** – How can we effectively measure the capabilities of LLMs at modelling parallel and HPC code?
Collecting a Parallel and HPC Code Dataset
Collecting a Parallel and HPC Code Dataset

- Dataset Objectives
Collecting a Parallel and HPC Code Dataset

- Dataset Objectives
  - Large amounts of parallel and HPC code from disparate sources and projects
Collecting a Parallel and HPC Code Dataset

- Dataset Objectives
  - Large amounts of parallel and HPC code from disparate sources and projects
  - Quality code data
Collecting a Parallel and HPC Code Dataset

- Dataset Objectives
  - Large amounts of parallel and HPC code from disparate sources and projects
  - Quality code data
  - Clean data; no duplicate files, no auto-generated code
Collecting a Parallel and HPC Code Dataset

- **Dataset Objectives**
  - Large amounts of parallel and HPC code from disparate sources and projects
  - Quality code data
  - Clean data; no duplicate files, no auto-generated code

- **HPC-Coder Dataset**
Collecting a Parallel and HPC Code Dataset

- **Dataset Objectives**
  - Large amounts of parallel and HPC code from disparate sources and projects
  - Quality code data
  - Clean data; no duplicate files, no auto-generated code

- **HPC-Coder Dataset**
  - Scrape GitHub for HPC repos with ≥ 3 stars
Collecting a Parallel and HPC Code Dataset

● Dataset Objectives
  ○ Large amounts of parallel and HPC code from disparate sources and projects
  ○ Quality code data
  ○ Clean data; no duplicate files, no auto-generated code

● HPC-Coder Dataset
  ○ Scrape GitHub for HPC repos with ≥ 3 stars
  ○ Filter by C/C++ source files
Collecting a Parallel and HPC Code Dataset

● **Dataset Objectives**
  ○ Large amounts of parallel and HPC code from disparate sources and projects
  ○ Quality code data
  ○ Clean data; no duplicate files, no auto-generated code

● **HPC-Coder Dataset**
  ○ Scrape GitHub for HPC repos with ≥ 3 stars
  ○ Filter by C/C++ source files
  ○ De-duplication by SHA-256 hash
Collecting a Parallel and HPC Code Dataset

- **Dataset Objectives**
  - Large amounts of parallel and HPC code from disparate sources and projects
  - Quality code data
  - Clean data; no duplicate files, no auto-generated code

- **HPC-Coder Dataset**
  - Scrape GitHub for HPC repos with ≥ 3 stars
  - Filter by C/C++ source files
  - De-duplication by SHA-256 hash
  - Remove large (> 1MB) and small (< 15 tokens) files
## Collecting a Parallel and HPC Code Dataset

<table>
<thead>
<tr>
<th>Filter</th>
<th># Files</th>
<th># LOC</th>
<th>Size (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>239,469</td>
<td>61,585,704</td>
<td>2.02</td>
</tr>
<tr>
<td>Deduplicate</td>
<td>198,958</td>
<td>53,043,265</td>
<td>1.74</td>
</tr>
<tr>
<td>Deduplicate + remove small/large files</td>
<td>196,140</td>
<td>50,017,351</td>
<td>1.62</td>
</tr>
</tbody>
</table>
## Collecting a Parallel and HPC Code Dataset

<table>
<thead>
<tr>
<th>Filter</th>
<th># Files</th>
<th># LOC</th>
<th>Size (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>239,469</td>
<td>61,585,704</td>
<td>2.02</td>
</tr>
<tr>
<td>Deduplicate</td>
<td>198,958</td>
<td>53,043,265</td>
<td>1.74</td>
</tr>
<tr>
<td>Deduplicate + remove small/large files</td>
<td>196,140</td>
<td>50,017,351</td>
<td>1.62</td>
</tr>
</tbody>
</table>

Approximately 18% of files are removed during preprocessing.
### Collecting a Parallel and HPC Code Dataset

<table>
<thead>
<tr>
<th>Filter</th>
<th># Files</th>
<th># LOC</th>
<th>Size (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>239,469</td>
<td>61,585,704</td>
<td>2.02</td>
</tr>
<tr>
<td>Deduplicate</td>
<td>198,958</td>
<td>53,043,265</td>
<td>1.74</td>
</tr>
<tr>
<td>Deduplicate + remove small/large files</td>
<td>196,140</td>
<td>50,017,351</td>
<td>1.62</td>
</tr>
</tbody>
</table>

![Distribution of Lines-of-Code by File Type](chart.png)
Focus on LLMs that fit on consumer GPUs
Selecting LLMs to Fine-Tune

- Focus on LLMs that fit on consumer GPUs
- Choose from a variety of pre-training data
Selecting LLMs to Fine-Tune

- Focus on LLMs that fit on consumer GPUs
- Choose from a variety of pre-training data
- Choose state-of-the-art LLMs in these categories (at the time)
Selecting LLMs to Fine-Tune

- Focus on LLMs that fit on consumer GPUs
- Choose from a variety of pre-training data
- Choose state-of-the-art LLMs in these categories (at the time)

<table>
<thead>
<tr>
<th>Model Name</th>
<th>No. of Parameters</th>
<th>Pre-Training Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-2</td>
<td>1.5B</td>
<td>natural language</td>
</tr>
<tr>
<td>GPT-Neo</td>
<td>2.7B</td>
<td>natural language + code</td>
</tr>
<tr>
<td>PolyCoder</td>
<td>2.7B</td>
<td>code</td>
</tr>
</tbody>
</table>
Fine-Tuning Methodology

- Fine-tune the existing LLMs on our dataset
- Auto-regressive fine-tuning
Fine-Tuning Methodology

- Fine-tune the existing LLMs on our dataset
- Auto-regressive fine-tuning

```c
int i = 0;
#pragma omp parallel ___
```
Fine-Tuning Methodology

- Fine-tune the existing LLMs on our dataset
- Auto-regressive fine-tuning

```c
int i = 0;
#pragma omp parallel tokens
#pragma omp parallel
```
Fine-Tuning Methodology

- Fine-tune the existing LLMs on our dataset
- Auto-regressive fine-tuning

```c
int i = 0;
#pragma omp parallel
```

LLM
Fine-Tuning Methodology

- Fine-tune the existing LLMs on our dataset
- Auto-regressive fine-tuning

```c
int i = 0;
#pragma omp parallel
```

```latex
\begin{align*}
    a & \quad 0.01 \\
    b & \quad 0.00 \\
    \vdots & \quad \vdots \\
    \text{for} & \quad 0.99 \\
    \vdots & \quad \vdots 
\end{align*}
```
Fine-Tuning Methodology

- Fine-tune the existing LLMs on our dataset
- Auto-regressive fine-tuning

```c
int i = 0;
#pragma omp parallel
```

![Diagram](image)
Fine-Tuning Methodology

- Fine-tune the existing LLMs on our dataset
- Auto-regressive fine-tuning
- Fine-tune for 1 epoch
- Record perplexity
  - Inversely proportional to how “perplexed” the LLM is by tokens in the distribution
  - Lower is better
- Run downstream tasks every 1000 steps
# Fine-Tuning Results

<table>
<thead>
<tr>
<th>Model</th>
<th>GPT-2</th>
<th>GPT-Neo</th>
<th>PolyCoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final Validation Perplexity</td>
<td>4.47</td>
<td>2.23</td>
<td>2.24</td>
</tr>
</tbody>
</table>
## Fine-Tuning Results

The larger models train to a lower perplexity.

<table>
<thead>
<tr>
<th>Model</th>
<th>GPT-2</th>
<th>GPT-Neo</th>
<th>PolyCoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final Validation Perplexity</td>
<td>4.47</td>
<td>2.23</td>
<td>2.24</td>
</tr>
</tbody>
</table>
The larger models train to a lower perplexity.

<table>
<thead>
<tr>
<th>Model</th>
<th>GPT-2</th>
<th>GPT-Neo</th>
<th>PolyCoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final Validation Perplexity</td>
<td>4.47</td>
<td>2.23</td>
<td>2.24</td>
</tr>
</tbody>
</table>
## Fine-Tuning Results

The larger models train to a lower perplexity.

<table>
<thead>
<tr>
<th>Model</th>
<th>Final Validation Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-2</td>
<td>4.47</td>
</tr>
<tr>
<td>GPT-Neo</td>
<td>2.23</td>
</tr>
<tr>
<td>PolyCoder</td>
<td>2.24</td>
</tr>
</tbody>
</table>

After ~45k steps the downstream performance drops, but perplexity keeps getting better.

![Evaluation Performance During Training](chart.png)

- average_pass@1
- average_pass@10
- average_pass@100
- Eval Perplexity
Evaluation Task 1: Code Generation

- How well can the LLMs generate code?
Evaluation Task 1: Code Generation

- How well can the LLMs generate code?
- 25 unique kernels spanning serial, OpenMP, and MPI code
Evaluation Task 1: Code Generation

- How well can the LLMs generate code?
- 25 unique kernels spanning serial, OpenMP, and MPI code
- Measure the pass@k
How well can the LLMs generate code?

- 25 unique kernels spanning serial, OpenMP, and MPI code
- Measure the pass@k

```c
/* Compute the dot product of x and y using OpenMP */
int product(int *x, int *y, size_t N) {
```
Evaluation Task 1: Code Generation

- How well can the LLMs generate code?
- 25 unique kernels spanning serial, OpenMP, and MPI code
- Measure the pass@k

```c
/* Compute the dot product of x and y using OpenMP */
int product(int *x, int *y, size_t N) {
```
Evaluation Task 1: Code Generation

- How well can the LLMs generate code?
- 25 unique kernels spanning serial, OpenMP, and MPI code
- Measure the pass@k

```c
/* Compute the dot product of x and y using OpenMP */
int product(int *x, int *y, size_t N) {
    int sum = 0;
    #pragma omp parallel for
    for (int i = 0; i < N; i++) {
        sum += x[i] * y[i];
    }
    return sum;
}
```
Evaluation Task 1: Code Generation

- How well can the LLMs generate code?
- 25 unique kernels spanning serial, OpenMP, and MPI code
- Measure the pass@k

```c
/* Compute the dot product of x and y using OpenMP */
int product(int *x, int *y, size_t N) {
    int sum = 0;
    for (int i = 0; i < N; i++) {
        sum += x[i] * y[i];
    }
    return sum;
}
```
**Evaluation Task 1: Code Generation**

- How well can the LLMs generate code?
- 25 unique kernels spanning serial, OpenMP, and MPI code
- Measure the pass@k

```c
/* Compute the dot product of x and y using OpenMP */
int product(int *x, int *y, size_t N) {
```
Evaluation Task 1: Code Generation

● How well can the LLMs generate code?
● 25 unique kernels spanning serial, OpenMP, and MPI code
● Measure the pass@k

/* Compute the dot product of x and y using OpenMP */
int product(int *x, int *y, size_t N) {

prompt
LLM

response 1
response 2
::
response k

What is the probability at least one of k responses is correct?
Evaluation Task 1: Code Generation

- How well can the LLMs generate code?
- 25 unique kernels spanning serial, OpenMP, and MPI code
- Measure the pass@k

```
/* Compute the dot product of x and y using OpenMP */
int product(int *x, int *y, size_t N) {
```
Evaluation Task 1: Code Generation

HPC tuned models perform much better than PolyCoder.
Evaluation Task 2: OpenMP Pragma Generation

- Fine-tune LLMs to predict OpenMP pragmas
- Create dataset of 16k for loops from earlier dataset
- Fine-tune for 3 epochs
Evaluation Task 2: OpenMP Pragma Generation

- Fine-tune LLMs to predict OpenMP pragmas
- Create dataset of 16k for loops from earlier dataset
- Fine-tune for 3 epochs

```c
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    x[i] = foo(x[i]);
}
```
Evaluation Task 2: OpenMP Pragma Generation

- Fine-tune LLMs to predict OpenMP pragmas
- Create dataset of 16k for loops from earlier dataset
- Fine-tune for 3 epochs

```c
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    x[i] = foo(x[i]);
}
```

```c
for (int i = 0; i < N; i++) {
    x[i] = foo(x[i]);
}
```

<OMP>
```c
#pragma omp parallel for
```
```c
END_OMP
```
Evaluation Task 2: OpenMP Pragma Generation

Up to 97% accuracy predicting the OpenMP pragmas.
Evaluation Task 3: Relative Performance Modeling

- Compile and run entire commit history of Kripke and Laghos
- Fine-tune LLM as classifier to predict performance degradation given commit diff
- 1 – performance improved or stayed the same; 0 – performance got worse
Evaluation Task 3: Relative Performance Modeling

Up to 92% accuracy predicting performance regressions.
Conclusion and Takeaways

- Fine-tuning can improve the performance of code LLMs on low data resource problems
- State-of-the-art LLMs are bad at parallel and HPC tasks
- We need custom evaluations on HPC and parallel tasks
Contributions and Next Steps

- A large, HPC source code dataset
- A fine-tuned HPC code LLM: HPC-Coder
- Benchmarks for evaluating LLMs on HPC tasks
- HPC-Coder-v2 in coming weeks…

“Can Large Language Models Write Parallel Code?” HPDC ‘24

“Performance-Aligned LLMs for Generating Fast Code” arXiv 2404.18864

dnicho@umd.edu