Multi Agent System for Local LLM-Based HPC Code Generation

Ryo Mikasa[1], Daichi Mukunoki[2], Koki Morita[3], Shun-Ichirou Hayashi[3], Tetsuya Hoshino[2], Takahiro Katagiri[2]

1 School of Informatics, Nagoya University, 2 Information Technology Center, Nagoya University
3 Graduate School of Informatics, Nagoya University

1 Introduction

◆**Background**

- ✓Large Language Models (LLMs) have been increasingly used for code generation, including High-Performance Computing (HPC) applications
- ✓LLM-based agents can automate code generation and code testing and validation

◆**Problem**

- ✓Most existing agent frameworks rely on closed-source LLM APIs
  - This raises **security and privacy** concerns when handling confidential code or proprietary software
- ✓Using locally deployed LLMs is a potential solution However, when directly integrated into existing frameworks[1], their performance is often suboptimal

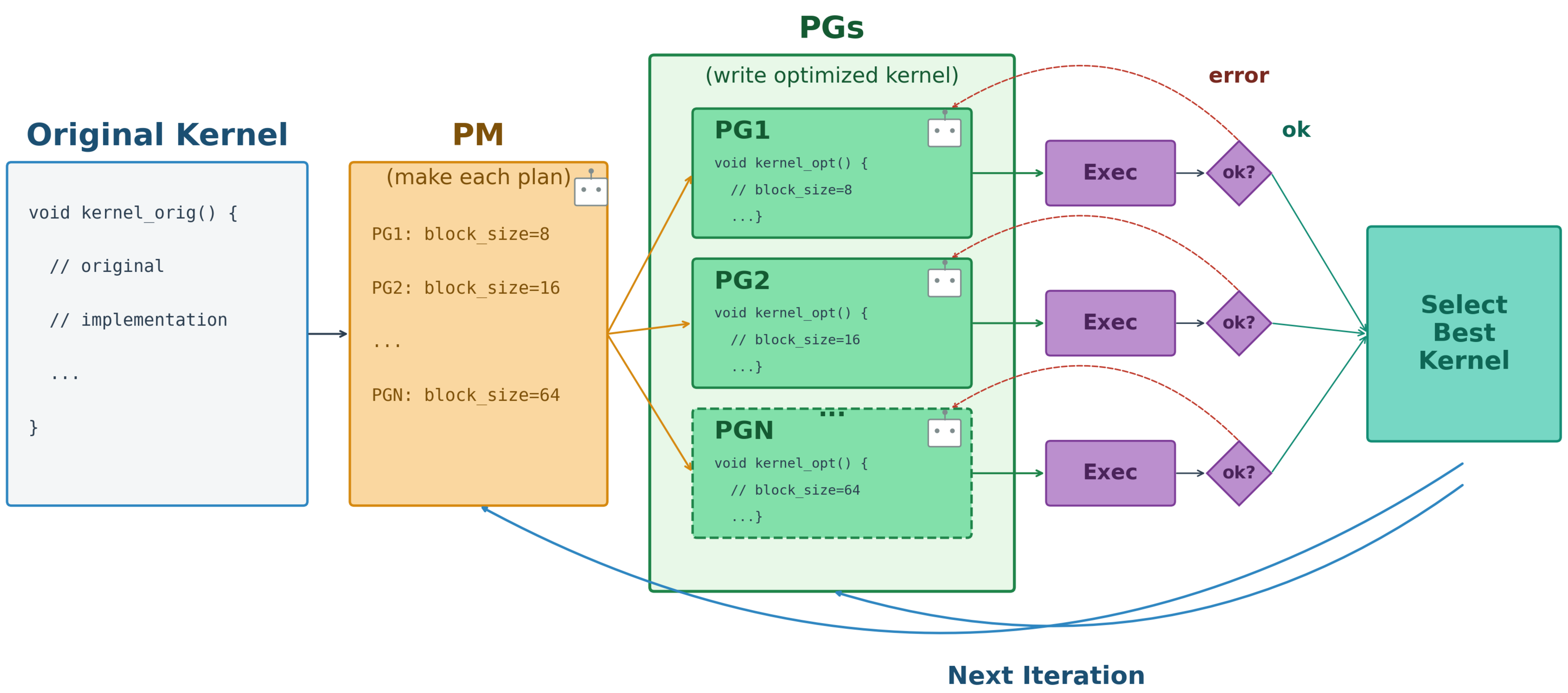
◆**Proposed Approach**

- ✓ Supports parallel inference across multiple LLM servers
  - Enables efficient utilization of computational resources
  - Reduces latency compared to conventional sequential agent systems
- ✓ Achieves faster and more efficient code generation for HPC applications

◆**Key Features and Benefits**

- ✓ We propose a prototype agent system specifically designed to
  - operate reliably with local LLMs
  - improve performance in HPC code generation tasks
- ✓ The system investigates
  - architectural designs that enhance the effectiveness of local LLM-based agents

2 System Design



◆ **Role of LLM agents**

- ✓ PM (Project Manager)
  - Input: The currently best-performing kernel
  - Output: Plans optimization strategies for each PG
- ✓ PGs (Programmers)
  - Input: PM-planned strategy (orange line) or Previously created code and the resulting error (red line)
  - Output: Optimized kernel

◆ **Iterative Process**

1 PM assigns different strategies to each PG
2 PGs implement optimized kernels in parallel
3 Execute and validate each kernel
4 If an error occurs, the programmer rewrites the code based on the error details
5 Select fastest valid kernel → Next iteration

◆ **Constraint**

- The agent edits only a single source file; compiler flags and build scripts are fixed (cannot be modified)

3 Evaluation

◆**Experimental Conditions**

- ✓ **Hardware:** Intel Xeon Gold 6230 × 2 (40 cores total), 2688 TFlops FP64, 2815 GB/s memory bandwidth
- ✓ **Compiler:** GCC 1130 with -O3 -march=native –fopenmp
- ✓ **LLM:** gpt-oss-120b[2]
- ✓ **Measurement:** Best of 5 executions per kernel
- ✓ **Reference:** Intel MKL 202320 for correctness validation and baseline
- ✓ **Comparison target:** performance differences at 1-, 2-, 4-, and 8-way parallelism of PGs

◆**Result**

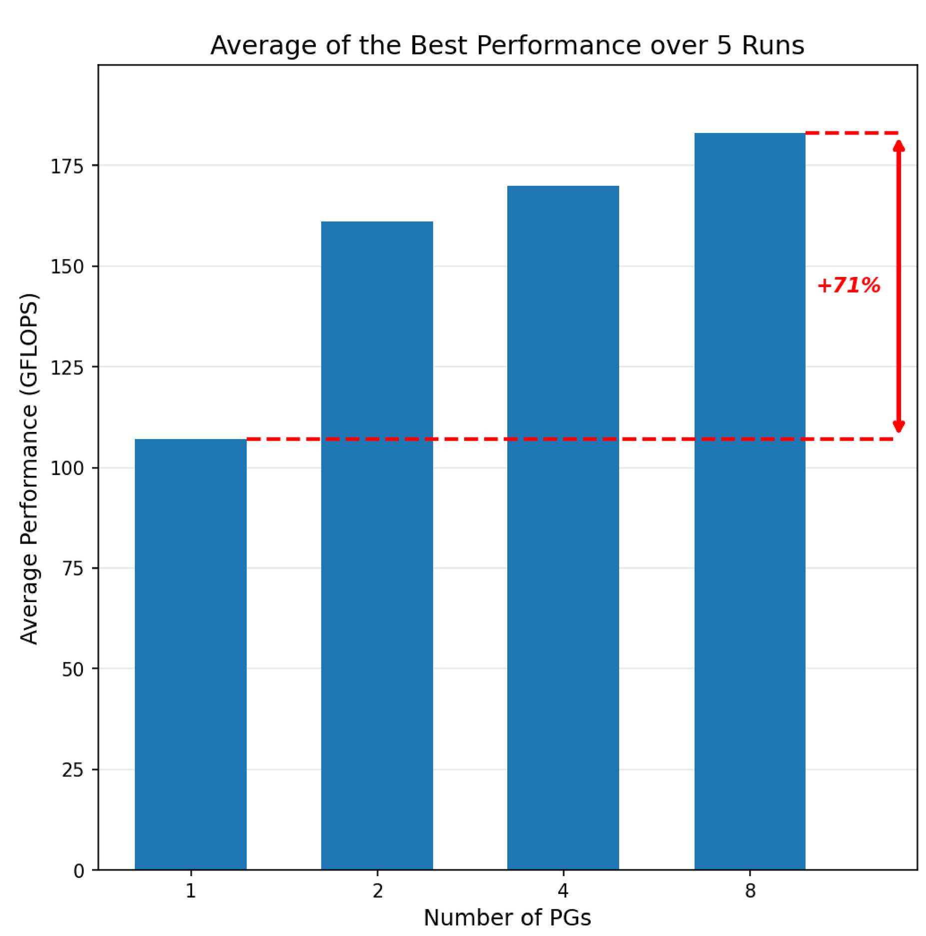
- ✓ **Ability of LLM :**
  - gpt-oss-120b has fundamental knowledge of optimization techniques in the HPC code domain, including correct usage of OpenMP directives and SIMD intrinsic instructions
  - It also understands performance optimization methods such as blocking, memory prefetching, and data alignment
  - By effectively eliciting this knowledge, it can be applied directly to practical code implementation and performance tuning
- ✓ **Effects of Parallelization**
  - Increasing the number of PGs results in higher peak performance Notably, when comparing parallelism levels 1 to 8, we observe a 71% performance improvement
  - By running more PGs in parallel, the total number of code generation attempts across the system increases, improving the chances of discovering better-performing code

Original kernel (094 GFLOPS)

```c
void matmul_original(int N, double *A, double *B, double *C) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            double sum = 0.0;
            for (int k = 0; k < N; k++) {
                sum += A[i * N + k] * B[k * N + j];
            }
            C[i * N + j] = sum;
        }
    }
}
```

Best kernel in this experiment (Number of PGs is 8, 35491 GFLOPS)

```c
void matmul_optimized(int N, double *A, double *B, double *C) {
    const int BM = 64;
    const int BN = 64;
    const int BK = 64;
    #pragma omp parallel for collapse(2) schedule(static)
    for (int i0 = 0; i0 < N; i0 += BM) {
        for (int j0 = 0; j0 < N; j0 += BN) {
            for (int k0 = 0; k0 < N; k0 += BK) {
                int imax = (i0 + BM > N) ? N - i0 : BM;
                int jmax = (j0 + BN > N) ? N - j0 : BN;
                int kmax = (k0 + BK > N) ? N - k0 : BK;
                double *packA;
                double *packB;
                posix_memalign((void **)&packA, 64, BM * BK * sizeof(double));
                posix_memalign((void **)&packB, 64, BK * BN * sizeof(double));
                for (int ii = 0; ii < imax; ++ii) {
                    for (int kk = 0; kk < kmax; ++kk) {
                        packA[ii * BK + kk] = A[(i0 + ii) * N + (k0 + kk)];
                    }
                }
                for (int kk = 0; kk < kmax; ++kk) {
                    for (int jj = 0; jj < jmax; ++jj) {
                        packB[kk * BN + jj] = B[(k0 + kk) * N + (j0 + jj)];
                    }
                }
                for (int ii = 0; ii < imax; ++ii) {
                    for (int jj = 0; jj < jmax; jj += 8) {
                        __m512d cvec = _mm512_loadu_pd(&C[(i0 + ii) * N + (j0 + jj)]);
                        for (int kk = 0; kk < kmax; ++kk) {
                            __m512d avec = _mm512_set1_pd(packA[ii * BK + kk]);
                            __m512d bvec = _mm512_loadu_pd(&packB[kk * BN + jj]);
                            cvec = _mm512_fmadd_pd(avec, bvec, cvec);
                        }
                        _mm512_storeu_pd(&C[(i0 + ii) * N + (j0 + jj)], cvec);
                    }
                }
                free(packA);
                free(packB);
            }
        }
    }
}
```



4 Conclusion

◆In this work, we developed a framework for an iterative HPC code optimization system using local large language models

◆In this study, we observed that increasing the level of parallel code generation tends to improve the quality of the generated code In particular, when using local LLMs in a personal computing environment, parallel inference can be an effective approach for better utilizing available machine resources

◆Although the results are promising, the framework is still under development and requires further refinement and evaluation

Acknowledgements

This work was supported by the Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures (JHPCN) and the High Performance Computing Infrastructure (HPCI) under Project ID: jh250015, and by JSPS KAKENHI Grants JP23K11126 and JP24K02945

References

[1] Dong Huang, Qingwen BU, Jie M Zhang, Michael Luck, Yuhao Qing, andHeming Cui 2024 AgentCoder: Multi-Agent Code Generation with EffectiveTesting and Self-optimisation arXiv preprint arXiv:231213010v3 (may 2024)arXiv:231213010v3 [csCL] Version v3 (24 May 2024)

[2] OpenAI 2025 gpt-oss-120b & gpt-oss-20b Model Card arXiv:250810925 [csCL]https://arxivorg/abs/250810925