LLM-Assisted GPU Kernel Generation

UTD RIDE Program · Sep - Dec 2025

1 · From Large to Small: Transferring CUDA Optimization Expertise via Reasoning Graph

Idea

ReGraph - based iterative refinement would generate and refine CUDA code based on real-world scenarios and latest performance reports.

Main Components

Main components would be taking in sequential code and target device requirements/restrictions. The loop would first load ReGraph into its system where an LLM or MCGS would use ReGraph to search for the best chain of optimization steps to generate the CUDA kernel. After compiling/running the generated kernel on the target device, cases will be tested to ensure all functionalities are correct. To track performance, we can use a reward signal (similar to the reward system of CUDAForge). The generated report would then be fed back into the LLM/MCGS agent to improve the process for the next iteration.

Why This Design Would Outperform Existing Approaches

This design would outperform existing approaches because it addresses the multi-step nature of the process and uses the reward system. While using the basic LLM strategy needs a series of interdependent steps, ReGraph would iterate optimizations, meaning the generation process repeats multistep optimization techniques to decrease risk of flawed kernel generation. Additionally, existing approaches rely on passing basic test cases, which is just a synthetic metric. The new design will use a performance profiler as the reward system since the optimized kernel generation process is a performance based task. The "reward system" is the measured speed increase on the target hardware after the self-evaluation process. Using MCGS (as discussed in the paper), the performance metric will update Q-values or visit counts, making it easier to track the speed improvements. This ensures the agent's optimization technique is based on logistical reasoning rather than predictions or memorization.

Pseudocode

```
FUNCTION ReGraph_ClosedLoop_Kernel_Generation(SequentialCode, TestCases, MaxIteration):

  // Initialization.
  // Load ReGraph onto system.
  BaseRuntime = Profile_Execution(SequentialCode)
  BestKernel  = SequentialCode
  BestSpeedup = 1.0

  FOR iteration = 1 TO MaxIterations DO:

    // Kernel Generation.
    TargetKernel = MCGS_Search(SequentialCode, ReGraph, BestSpeedup)
    // The search will navigate ReGraph to apply optimization sequence.

    // Compilation + Verification.
    CompilationSuccess = Compile(CandidateKernel)
    IF CompilationSuccess = true:
      CorrectOutput = Verify_Output(TargetKernel, TestCases)
      IF CorrectOutput:

        // Performance Profiler (Reward System).
        CurrentRuntime  = Profile_Execution(TargetKernel)
        CurrentSpeedup  = BaseRuntime / CurrentRuntime

        // Refinement + Update.
        IF CurrentSpeedup > BestSpeedup:
          BestSpeedup = CurrentSpeedup
```

```
        BestKernel  = TargetKernel
        Feedback    = "SUCCESS: New best kernel found (Speedup: + CurrentSpeedup)"
      ELSE:
        Feedback    = "MINOR: correct but slower/same speed (Speedup: + CurrentSpeedup)"
    ELSE:
      Feedback       = "FAILURE: Functionality correctness did not pass check."
      CurrentSpeedup = 0.0  // Negative reward signal.
  ELSE:
    Feedback         = "FAILURE: Compilation failed with error: " + CompileLog
    CurrentSpeedup   = 0.0  // Negative reward signal

  // Refinement + teaching.
  MCGS_Update_Policy(TargetKernel, CurrentSpeedup, Feedback)
  // Update ReGraph's edge weights/MCGS policy to adjust optimization sequence
  // that resulted in this.

RETURN BestKernel
```

2 · CUDAForge: An Agent Framework with Hardware Feedback for CUDA Kernel Optimization

Basic Idea

Using hardware data as feedback to guide LLM.

Main Components

Main components would include giving the LLM a high-stakes task and hardware specs to generate the initial CUDA code to be compiled for the target hardware. It then will execute the compiled binary on the hardware to retrieve performance metrics such as runtime, memory bandwidth etc. From there, test cases for correctness will be executed against the kernel (can also give a pass or fail flag). This agent will be trained and smaller than an LLM and it will take in the performance stats and test case results. It can then compare the results with a predeclared reward model (goal: minimum run time and maximum results). Based on the gap, the results can be translated to actionable steps for the generator to act on.

Why This Design Could Outperform Existing Approaches

This design could outperform existing approaches because of its hardware-specific feedback. Current methods rely on LLM's internal knowledge base, which limits the scope. This design uses hardware feedback (profiling data) to adjust the code. This allows the system to identify elusive optimization techniques that static analysis would not identify with such precision and speed. The feedback agent doesn't just check for errors, but it will actually lead the kernel to optimization through actionable steps. Additionally, the design will convert vague performance details to specific directions which will make the prompts much more specific and easy for the LLM to execute.

Pseudocode

```
FUNCTION OptimizedKernel_ClosedLoop_Design(TaskDescription, TargetHardware, PerformanceGoal)

  KernelCode                = NULL
  BestKernelCode            = NULL
  LastRefinementInstructions = "Generate initial CUDA kernel for: " + TaskDescription
  BestPerformanceMetric     = Infinity
  MAX_ITERATIONS            = 10
  Iteration                 = 0

  WHILE (Iteration < MAX_ITERATIONS) AND (BestPerformanceMetric > PerformanceGoal) DO

    // Kernel generation and compile.
    // LLM generates kernel based on feedback or prompt.
    KernelCode        = LLM_GENERATE(LastRefinementInstructions, BestKernelCode)
    CompilationResult = COMPILE(KernelCode, TargetHardware)

    IF CompilationResult.Success == FALSE THEN
      // Give error feedback to repair.
      LastRefinementInstructions = "Fix compilation error: " + CompilationResult.ErrorMessage
      Iteration = Iteration + 1
```

```
      CONTINUE
    END IF

    // Execute + profile kernel on target hardware.
    ExecutionMetrics         = EXECUTE_AND_PROFILE(CompilationResult.Executable, TargetHardware)
    VerificationResult       = RUN_TESTS(CompilationResult.Executable, TaskDescription)
    CurrentPerformanceMetric = ExecutionMetrics.Runtime

    // Feedback
    IF VerificationResult.Correctness == FALSE THEN
      // Correctness is main priority
      RawFeedback = "ERROR: Kernel failed verification. Failure details: " +
                    VerificationResult.Details
    ELSE IF CurrentPerformanceMetric < BestPerformanceMetric THEN
      // Kernel is correct AND improved performance.
      BestPerformanceMetric = CurrentPerformanceMetric
      BestKernelCode        = KernelCode
      RawFeedback = "SUCCESS: New best performance achieved. Runtime: " +
                    CurrentPerformanceMetric +
                    ". Analyze profiling data for further gains."
    ELSE
      // Kernel is correct but no performance improvement.
      RawFeedback = "CORRECT: Current Runtime: " + CurrentPerformanceMetric +
                    ". Goal: " + PerformanceGoal +
                    ". Analyze profile to identify bottleneck."
    END STATEMENT IF

    // Agent translates vague feedback into actionable directions
    LastRefinementInstructions = FEEDBACK_AGENT_GENERATE_PROMPT(KernelCode,
                                   RawFeedback, ExecutionMetrics)
    Iteration = Iteration + 1

  END WHILE

  RETURN BestKernelCode

END FUNCTION
```

3 · TritonGym: A Benchmark for Agentic LLM Workflows in Triton GPU Code Generation

Idea + Components

The architecture will rely on LLM agent to generate initial kernel and refine it iteratively based on feedback. Triton environment will compile and run the code, which will represent the primary source of feedback for LLM to use. The functional verification will be performed, checking to see whether the correct code was generated or not. Performance will also be profiled to track metrics. The reward model is a crucial component in this architecture because it will aggregate the raw execution data into a structured reward pointer as well as NL feedback for the LLM.

Why This Design Would Outperform Existing Approaches

Most LLM-assisted code generation only makes one attempt at generation or relies on manual updates to correct failed verification checkpoints, which is not efficient. This design will rather give LLM immediate feedback on why exactly the kernel failed, along with how it performed. The LLM can then refine itself iteratively, ensuring improvements are consistent and performance is not stagnated or lacking. By learning the ability to self-evaluate its generated code, the LLM familiarizes itself with optimization techniques used in Triton, resulting in stronger kernel generation and better performance that existing approaches.

Pseudocode

```
FUNCTION LLM_Kernel_Optimizer(User_Request, Max_Attempts, Performance_Target)

  Kernel_Code      = ""
  Attempt          = 0
  Best_Performance = Infinity
```

```
  Best_Kernel      = ""

  // main closed loop
  WHILE Attempt < Max_Attempts DO
    Attempt = Attempt + 1

    // Kernel generation
    IF Attempt == 1 THEN
      Kernel_Code = LLM.Generate_Initial_Kernel(User_Request)
    ELSE
      // LLM refines previous kernel using feedback
      Kernel_Code = LLM.Refine_Kernel(Kernel_Code, Feedback)
    END IF

    // Compile.
    Compilation_Result = Triton.Compile(Kernel_Code)
    IF Compilation_Result == ERROR THEN
      Feedback = "Compilation Failed: " + Compilation_Result.Error_Message
      Reward   = -10.0  // Heavy penalty for non-compiling code
    ELSE
      // Test and Profiling stage.
      Execution_Result = Triton.Execute_Kernel(Kernel_Code)
      IF Execution_Result.Correctness == FALSE THEN
        Feedback = "Functional Failure: Kernel produced incorrect output."
        Reward   = -5.0  // Penalty for incorrect logic
      ELSE
        // Execution succeeded and is correct
        Performance = Execution_Result.Latency

        // Feedback and Reward Stage
        IF Performance < Best_Performance THEN
          Best_Performance = Performance
          Best_Kernel      = Kernel_Code
        END IF

        // Reward function: Higher score for faster performance
        Reward   = max(0.0, 10.0 * (1.0 / Performance))
        Feedback = "Success. Latency: " + Performance + ". Target: " + Performance_Target

        IF Performance <= Performance_Target THEN
          RETURN Best_Kernel, "Optimized kernel found within target."
        END IF
      END IF
    END IF

    // Extra (to make it even more powerful) - update Reward Model
    // Can be done offline or online using state, action, reward tuple.
    Reward_Model.Update(Kernel_Code, Feedback, Reward)

  END WHILE

  IF Best_Kernel IS NOT EMPTY THEN
    RETURN Best_Kernel, "Best kernel after max attempts. Latency: " + Best_Performance
  ELSE
    RETURN NULL, "Failed to generate a functional kernel."
  END IF

END FUNCTION
```