Autotuning GPU Memory Access Patterns using SMT Solvers.

Saheed Olayemi Bolarinwa

Leibniz Supercomputing Centre (LRZ), Germany

saheed.bolarinwa@lrz.de

Introduction

Efficient memory access patterns are crucial for maximizing GPU performance. However, manually optimizing these patterns for specific algorithms and hardware architectures is time-consuming and requires deep expertise. Previous works like Jang et all[1], Chen et al[2], Ben van Werkhoven[3], Jo et al[4], Weber et al[5], Lim et al[6], and many others have explored empirical profiling strategies for automating this process. However, these methods are often limited in scalability and portability, they also require significant expert knowledge.

This work proposes using Satisfiability Modulo Theories (SMT) solvers to address these challenges. SMT solvers are powerful tools used to determine the satisfiability of logical formulas with respect to certain background theories[7]. They excel at handling complex constraints, making them invaluable in fields such as formal verification of hardware and software, model checking, optimization problems, synthesis, and artificial intelligence[8].

The challenge of tuning memory access patterns can be formulated as an optimization problem of minimizing memory access latency and maximizing throughput while considering constraints (e.g., cache hierarchy, coalesced access requirements) imposed by the target GPU architecture. This optimization problem can be modelled as a set of logical constraints given to SMT solvers to systematically and efficiently explore the vast space of potential configurations.

The proposed autotuning framework automatically determines optimal memory access pattern parameters (e.g., loop tiling factors, block sizes) for GPU kernels. The framework leverages SMT solvers to find configurations that offer the best performance on the given hardware.

Benefits

- **Scalability and Adaptability**: Efficiently handles large-scale and complex patterns, dynamically adapting to workload and hardware changes through real-time profiling and JIT compilation.
- **Accuracy**: Formal verification and real-time adjustments ensure optimal configurations.
- **Automation**: Reduces optimization time and effort significantly.
- **Portable Performance**: One-time optimization efforts applicable across various hardware with minimal code changes.
- **Uncovering Hidden Potential**: Discovers performance improvements that manual tuning might miss.

Methodology

Key Framework Components

1. **Applications/Kernels**: The target code for optimization, typically GPU-accelerated.
2. **Compiler Frameworks**: LLVM and MLIR offer reusable tools for optimizing code, generating machine code, and supporting various languages and architectures.
3. **SMT-Based Core**: uses SMT solvers for Constraint Modeling to formalize constraints and objectives from the compiler, Optimization to find the best configurations, and Verification to ensure correctness and feasibility.
4. **Dynamic Adaptation Mechanisms**: improves automation by employing techniques like Real-Time Profiling, and possibly Adaptive Learning Models.
5. **Automated Feedback Loop**: Close the optimization loop and apply improvement achieved continuosly.

Process Strategy

- ***Compiler Integration and Analysis (a,b)***
  - Extract GPU architectural features (e.g., memory hierarchy, bandwidth, cache sizes).
  - Analyze application code and instrument for profiling.
- ***SMT-Based Optimization (c)***
  - Formulates optimization problems as SMT constraints and objectives based on compiler analysis.
  - Encode these constraints in the SMT solver to find optimal configurations.
  - Verify the correctness of configurations and solve for optimal memory access patterns.
- ***Dynamic Adaptation (d,f)***
  - Continuously collects performance metrics using instrumented code.
  - Feed this data back into the SMT solver for dynamic re-optimization.
- ***Automated Feedback Loop (e,f,g)***
  - Close the optimization loop with real-time feedback.
  - Use JIT compilation for dynamic optimization based on real-time feedback

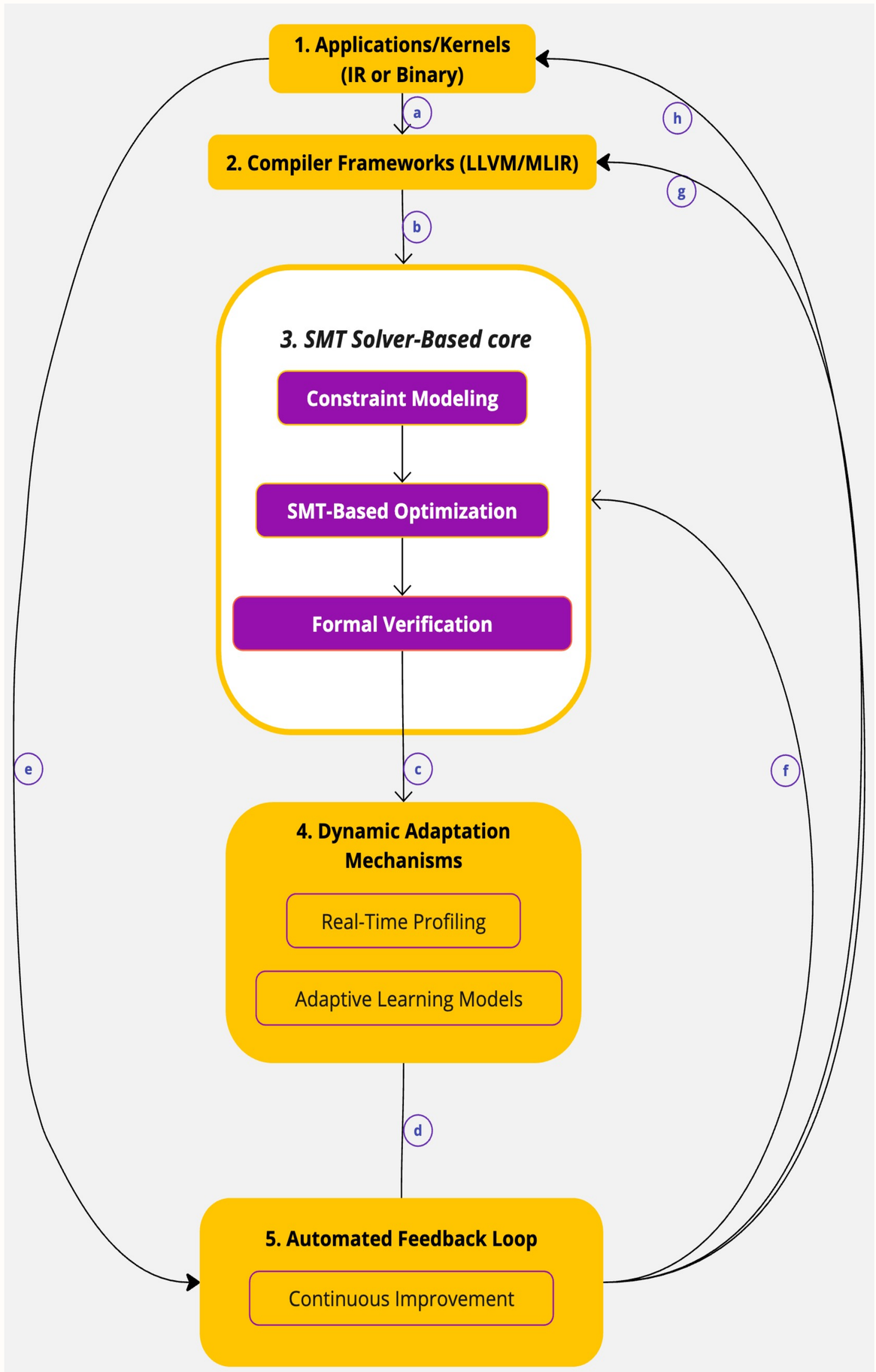

Figure 1: SMT based framework for autotunning GPU memory access.

Conclusion

By introducing SMT solvers into the process of autotuning, this research proposes a fundamentally new approach to GPU memory optimization. This holistic framework is designed to overcome the challenges of existing methods and deliver substantial performance improvements. The approach has the potential to open doors for further advancements in autotuning memory access patterns, not just for GPUs but for the increasingly diverse ecosystem of accelerators.

References

References

[1] B. Jang, D. Schaa, P. Mistry, and D. R. Kaeli, "Exploiting memory access patterns to improve memory perfor pp. 105–118, 2011.

[2] X. Chen, D. Su, Y. Wang, and H. Yang, "Nonzero pattern analysis and memory access optimization in gpu-ba *Applications: Architectures and Algorithms*, IA³ '13, (New York, NY, USA), Association for C

[3] B. van Werkhoven, "Kernel tuner: A search-optimizing gpu code auto-tuner," *Future Gener. Comput. Syst.*, v

[4] G. Jo, J. Jung, J. Park, and J. Lee, "Poster: Mapa: An automatic memory access pattern analyzer for gpu ap *Programming*, PPoPP '17, (New York, NY, USA), p. 443–444, Association for Computing Machinery, 2017.

[5] N. Weber, S. C. Amend, and M. Goesele, "Guided profiling for auto-tuning array layouts on gpus," in *Proceed Performance Computing Systems*, PMBS '15, (New York, NY, USA), Association for Computing Machinery, 2

[6] R. V. Lim, B. Norris, and A. D. Malony, "Autotuning gpu kernels via static and predictive analysis," *2017 46t*

[7] L. de Moura and N. Bjørner, "Z3: an efficient smt solver," in *2008 Tools and Algorithms for Construction and*

[8] D. Guidotti, L. Pandolfo, and L. Pulina, "Leveraging satisfiability modulo theory solvers for verification of neu