Visualization of Data Movements and Accesses

Seminar Thesis

Til Mohr

Chair for High Performance Computing, IT Center,
RWTH Aachen, Seffenter Weg 23,
52074 Aachen, Germany
Supervisor: Isa Thäringen, M.Sc.

This is the abstract. It is a short summary of the thesis contents (100 to 150 words).

Keywords: data locality, data movement, data access, optimization, visualization

1 Introduction

- Increasing Processor-Memory Speed Gap (2.1) and increasing requirements (2.2) result in a large increase in the affect of data movement costs and their resulting bottlenecks. While breakthroughs in hardware research can improve this issue, software engineers can also try to mitigate these issues by improving data locality (2.4) through software.
- Increasing complexity of programs makes it difficult to create a mental model of the data movement of a program. This makes it challenging for experts and impossible for domain researchers to optimize such a program.

2 Memory-Related Performance Problems

Perhaps: [17, 18]

2.1 Processor-Memory Speed Gap

Present the processor-memory speed gap. Explanation, why it is a problem will

2.2 Computation and Memory Requirements

Present that requirements are increasing rapidly based on examples like ChatGPT, etc. -> There is always a need for more memory and more computation power

2.3 Data Transfer Costs and Bottlenecks

Explain what common data transfer costs are (move from memory to caches and in between caches) and briefly explain bottlenecks (e.g. Cache misses, memory bandwidth, etc.)

2.4 Data Locality

Explain/Define the term data locality

3 Data Gathering and Visualization Approaches

The goal is to make it easily accessible and possible for everyone (experts and domain researchers) to understand a programs' data movements and fix issues. To do this, data has to be gathered automatically (3.1,3.2,3.3) and then visualized in an understandable manner (3.4).

In this section, we will discuss the different approaches to gathering data for visualizing data movements in a program.

3.1 Dynamic Analysis

Run program and gather data while it is running, using Hardware Counters, Profiling, or Tracing. [8, 14, 4, 13, 1]

Advantages:

- No need for parameterization → already is compiled for specific hardware

- Can be used in combination with actual data → even more accurate information

Disadvantages:

- Running a whole program is expensive (time and cost)
- Difficult to isolate and analyze specific parts of the program
- Very coarse time granularity → can not measure very short time intervals (hardware counters are not precise enough in aspects of being updated / read)

3.2 Static Analysis

Analyze the program statically using a compiler [16, 15, 3, 12, 11, 5, 2]

Advantages:

- Can be used to analyze specific parts of the program
- Fast and cheap → No need to run program

Disadvantages:

- Needs to be parameterized for specific hardware (and often not accurate enough → might miss some details)
- Can not be used in combination with actual data

3.3 Simulation

Simulate the program on a simulator [16, 9, 6, 10]

Advantages: **TODO**

- Can be used to analyze specific parts of the program
- In between Static and Dynamic Analysis in terms of precision and speed and cost

Disadvantages: **TODO**

3.4 Visualization Techniques

Very brief overview of different visualizations (Colored Graphs, Heatmaps, etc.)

4 Memory Access Visualization Tools

Take ~3 papers, briefly present how they work (which data gathering technique and what visualization), and what results they have shown.

4.1 MemAxes: Visualization and Analytics for Characterizing Complex Memory Performance Behaviors

[8]

4.2 Abstract Visualization of Runtime Memory Behavior

[6]

4.3 Boosting Performance Optimization with Interactive Data Movement Visualization

[16]

4.4 Comparison

5 Conclusions

Conclusion

Future Work:

- Data Gathering and Visualizations can always be improved
- But we can use the gathered data to automatically optimize programs [5] → Even less work for the programmer (who might be just a domain researcher)
- Deep Learning is also being experimented with to automatically optimize programs at compile time [7]

References

[1] Laksono Adhianto et al. "HPCToolkit: Tools for performance analysis of optimized parallel programs". In: *Concurrency and Computation: Practice and Experience* 22.6 (2010), pp. 685–701.

[2] Tal Ben-Nun et al. "Bridging Control-Centric and Data-Centric Optimization". In: *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization*. 2023, pp. 173–185.

[3] Tal Ben-Nun et al. "Stateful dataflow multigraphs: A data-centric model for performance portability on heterogeneous architectures". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2019, pp. 1–14.

[4] Abhinav Bhatele et al. “Novel views of performance data to analyze large-scale adaptive applications”. In: *SC’12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE. 2012, pp. 1–11.

[5] Alexandru Calotoiu et al. “Lifting C semantics for dataflow optimization”. In: *Proceedings of the 36th ACM International Conference on Supercomputing*. 2022, pp. 1–13.

[6] ANM Imroz Choudhury and Paul Rosen. “Abstract visualization of runtime memory behavior”. In: *2011 6th International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*. IEEE. 2011, pp. 1–8.

[7] Chris Cummins et al. “Programl: A graph-based program representation for data flow analysis and compiler optimizations”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 2244–2253.

[8] Alfredo Giménez et al. “Memaxes: Visualization and analytics for characterizing complex memory performance behaviors”. In: *IEEE transactions on visualization and computer graphics* 24.7 (2017), pp. 2180–2193.

[9] Julian Hammer et al. “Kerncraft: A tool for analytic performance modeling of loop kernels”. In: *Tools for High Performance Computing 2016: Proceedings of the 10th International Workshop on Parallel Tools for High Performance Computing, October 2016, Stuttgart, Germany*. Springer. 2017, pp. 1–22.

[10] Roman Iakymchuk and Paolo Bientinesi. “Modeling performance through memory-stalls”. In: *ACM SIGMETRICS Performance Evaluation Review* 40.2 (2012), pp. 86–91.

[11] Jeffrey Kodosky. “LabVIEW”. In: *Proceedings of the ACM on Programming Languages* 4.HOPL (2020), pp. 1–54.

[12] Stan Matwin and Tomasz Pietrzykowski. “Prograph: a preliminary report”. In: *Computer Languages* 10.2 (1985), pp. 91–126.

[13] Kathryn S McKinley and Olivier Temam. “Quantifying loop nest locality using SPEC’95 and the perfect benchmarks”. In: *ACM Transactions on Computer Systems (TOCS)* 17.4 (1999), pp. 288–336.

[14] Sergio Moreta and Alexandru Telea. “Visualizing dynamic memory allocations”. In: *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. IEEE. 2007, pp. 31–38.

[15] Philipp Schaad. “Boosting Performance Engineering with Visual Interactive Optimization and Analysis”. MA thesis. ETH Zurich, 2021.

[16] Philipp Schaad, Tal Ben-Nun, and Torsten Hoefler. “Boosting performance optimization with interactive data movement visualization”. In: *arXiv preprint arXiv:2207.07433* (2022).

[17] Adrian Tate et al. *Programming abstractions for data locality*. Tech. rep. Office of Scientific and Technical Information (OSTI), 2014.

[18] Didem Unat et al. “Trends in data locality abstractions for HPC systems”. In: *IEEE Transactions on Parallel and Distributed Systems* 28.10 (2017), pp. 3007–3020.