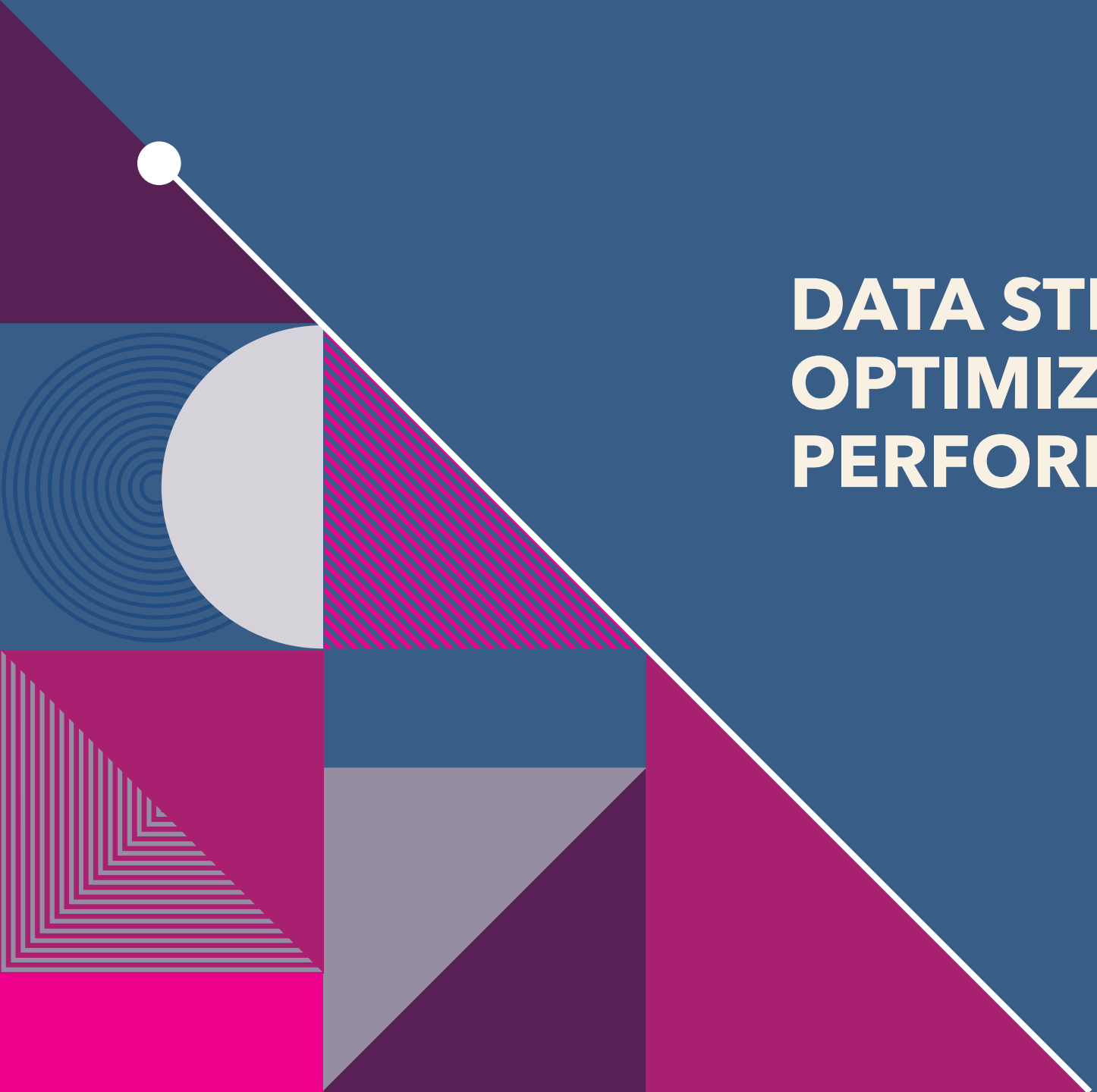


DATA STRUCTURE OPTIMIZATION IN HIGH-PERFORMANCE COMPUTING

ABSTRACT

•HPC applications require efficient data structures for maximum performance.

•Optimization replaces inefficient structures with cache-friendly alternatives.

•Focus on replacing linked lists with arrays for better memory locality.

•Python implementation showcases the performance differences.

INTRODUCTION

Importance of Data Structure Optimization in HPC

- Memory locality and cache utilization impact performance.
- Linked lists vs. arrays: performance trade-offs.

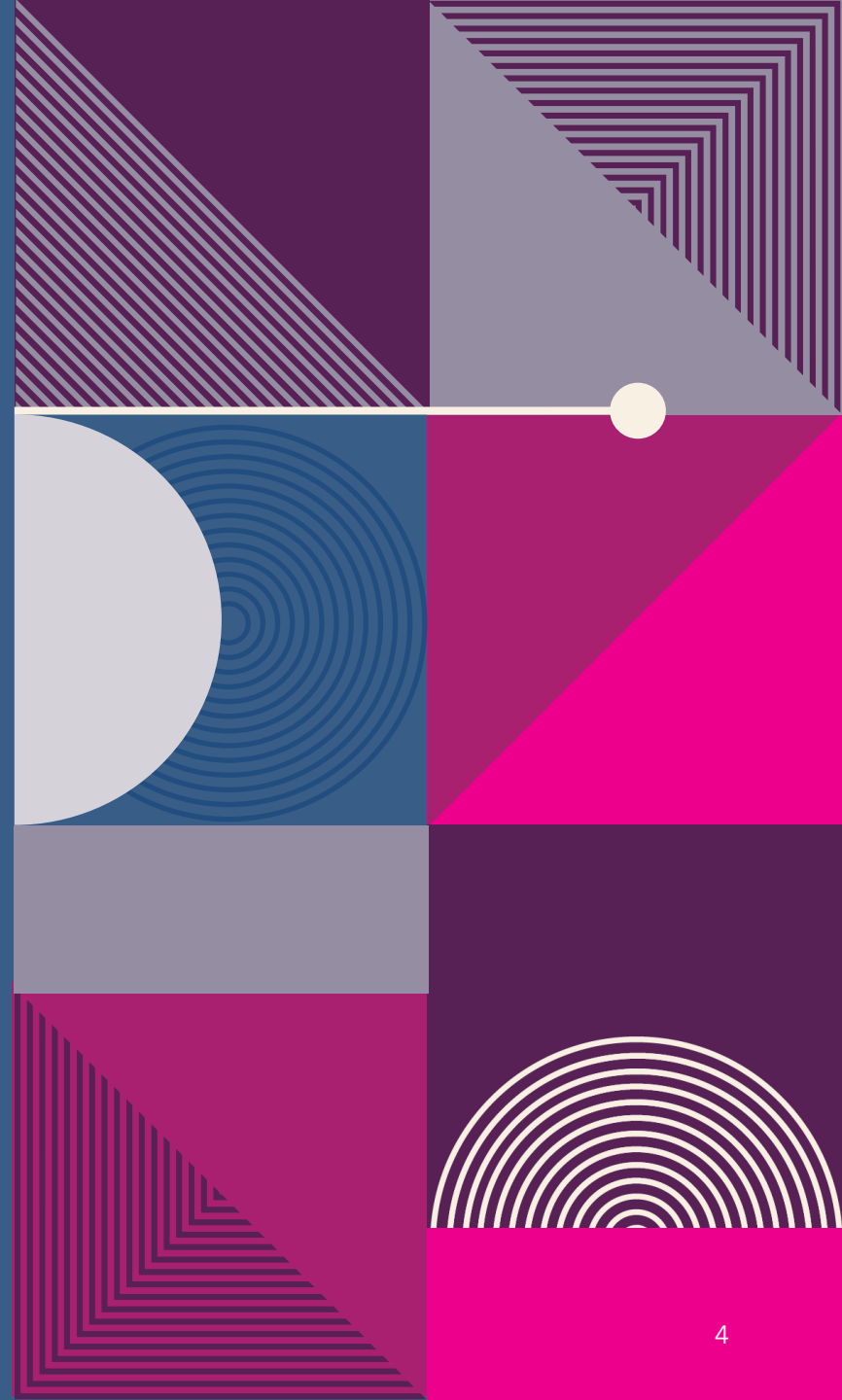
Research Justification

- Performance improvements demonstrated via prototype implementation.

OPTIMIZATION STRATEGY

Strengths of Data Structure Optimization:

- Improved cache locality.
- Reduced pointer overhead.
- Better parallelization capabilities.



WEAKNESSES OF DATA STRUCTURE OPTIMIZATION

- Fixed size constraint in arrays.
- High overhead for insertion/deletion in arrays.
- Need for hybrid structures for dynamic memory handling.

IMPLEMENTATION IN PYTHON

- **Problem Statement:**
  - Compare linked list vs. array performance using Python.
  - Calculated the sum of all the elements.
- **Testing Methodology:**
  - Data sizes: 100 to 100,000 elements.
  - 10 iterations per test for reliability.

- **Results Analysis:**
- Arrays perform better due to contiguous memory allocation and optimized C-level execution.
- Linked lists suffer from pointer dereferencing overhead.

LESSONS LEARNED

Key takeaways for HPC optimization:

- Favor contiguous memory structures when possible.
- Implement hybrid approaches for flexibility.
- Optimize memory management strategies.

IMPLICATIONS OF HPC

- Code refactoring to favor efficient data structures.
- Hybrid approaches combining arrays and linked lists.
- Efficient memory allocation to reduce system bottlenecks.
- Leveraging SIMD (Single Instruction Multiple Data) and GPU acceleration for HPC performance.

CONCLUSION

- Python implementation confirms benefits of arrays over linked lists.
- Data structure selection is crucial for HPC performance.
- Future research should explore adaptive data structures for workload-based optimization.

REFERENCES

Azad, M. A. K., Iqbal, N., Hassan, F., & Roy, P. (2023). "*An Empirical Study of High-Performance Computing (HPC) Performance Bugs*."

Tan, J., Jiao, S., Milind Chabbi, & Liu, X. (2020). *What every scientific programmer should know about compiler optimizations?* https://doi.org/10.1145/3392717.3392754

Su, P., Jiao, S., Milind Chabbi, & Liu, X. (2019). *Pinpointing performance inefficiencies via lightweight variance profiling*. 1–19. https://doi.org/10.1145/3295500.3356167

Frigo, M., & Johnson, S. G. (2005). The Design and Implementation of FFTW3. Proceedings of the IEEE, 93(2), 216–231. https://doi.org/10.1109/jproc.2004.840301

Ghorpade, J. (2012). GPGPU Processing in CUDA Architecture. *Advanced Computing: An International Journal*, *3*(1), 105–120. https://doi.org/10.5121/acij.2012.3109

Zhou, K., Meng, X., Sai, R., Dejan Grubisic, & Mellor-Crummey, J. (2021). An Automated Tool for Analysis and Tuning of GPU-Accelerated Code in HPC Applications. *IEEE Transactions on Parallel and Distributed Systems*, *33*(4), 854–865. https://doi.org/10.1109/tpds.2021.3094169