EFFECTS OF COMPILER OPTIMIZATIONS IN OPENMP TO CUDA TRANSLATION

Amit Sabne, Putt Sakdhnagool and Rudolf Eigenmann

Motivation

- **Our Goal:** To determine the impact of individual optimization techniques in the OpenMP to CUDA translator.

Evaluating the Impact of Optimization Techniques

- Bases our study on OpenMPC systems
- Uses method from Blume study[1]
  - Iteratively turns off one optimization at a time from the highest-optimized program variant
  - Measures the performance in term of slowdown incurred

How can we find the "best" combination of optimization techniques?

Finding Best Combination of Optimization Techniques

- The are 18 optimization techniques available in OpenMPC as compiler flags.

OpenMPC Optimization Techniques

Program Environment Configuration

- cudaThreadBlockSize=N
- assumeNonZeroTripLoop

Data Caching Strategy

- shrdSclrCachingOnReg
- shrdArryElmtCachingOnReg
- shrdSclrCachingOnSM
- prvtArryCachingOnSM
- shrdArryCachingOnTM
- shrdSclrCachingOnConst
- shrdArryCachingOnConst

Data Offloading Optimization

- useMallocPitch
- useGlobalGMalloc
- globalGMallocOpt
- cudaMallocOptLevel=N
- cudaMemTrOptLevel=N

Code Transformantion

- localRedVarConf=N
- useMatrixTranspose
- useParallelLoopSwap
- useUnrollingOnReduction

Finding Best Combination of Optimization Techniques

- Exhaustive Search
  - Search space is very large (2^n combinations for n on-off flags)
- Pruned Exponential Search (PE)
  - Used by OpenMPC tuning system
  - Use exhaustive search on pruned search space
    - The search space is reduced by using aggressive pruning heuristic.
  - Problems with PE
    - Search space is still large
    - Aggressive pruning heuristics may eliminate the best optimization combination

Finding Best Combination of Optimization Techniques

- Another Problem: **Runtime Variation**
  - Changes in the system that affect execution time of measuring program
- On GPUs Program, most of the variations come from memory transfer

Table 2. Variations on GPU Programs

Benchmark	Relative Standard Deviation for Memory Transfer Time (A)	Relative Standard Deviation for Computation Time (B)	Ratio (A/B)
NW (8192)	0.2395	0.0128	18.71
Jacobi (12288)	0.7394	0.0001	7394
CG (W)	0.2562	0.0706	3.63
FT (W)	0.1521	0.0112	13.58

NEW TUNING ALGORITHM

Modified Iterative Elimination (MIE)

- Based on Iterative elimination (IE) algorithm [2]
  - $O(n^2)$ Complexity
  - All optimization options are available at tuning time

For ***n*** optimization option {$F_1, F_2, \ldots, F_n$}

Let ***B*** = [$F_1, F_2, \ldots, F_n$] is the set of switched-on option

```
for i = 1 to n
        for F_j in B
                NextB = min_runtime(NextB, B - F_j) //Switches off F_j
        end for
        if NextB == B
                break
        end if
        B = NextB
end for
```

Modified Iterative Elimination (MIE)

- Dealing with runtime variation
  - Averaging the execution time of multiple runs is needed to reduce the effect of the variations
- On GPU program, memory transfer is the main source of runtime variation
  - Removes memory transfer time from execution time
    - Memory transfer behavior must be the same between comparing combination
  - Only optimizations that change memory transfer behavior need averaging across multiple

Modified Iterative Elimination (MIE)

- MIE separates tuning process into 4 phases

Phase 1
- Optimizations that affect memory transfer behavior
- Runs each IE stage multiple times and averages execution times for making decisions

Phase 2
- Optimizations that affect computation time
- Runs normal IE. Considers only computation time for making decision

Phase 3
- Optimizations that affect memory transfer behavior but depend on computation optimization

Phase 4
- Optimizations that are independent from first three phases

Modified Iterative Elimination (MIE)

- MIE separates tuning process into 4 phases

Phase 1
- Memory Transfer optimization
- All data offloading optimization except useMallocPitch

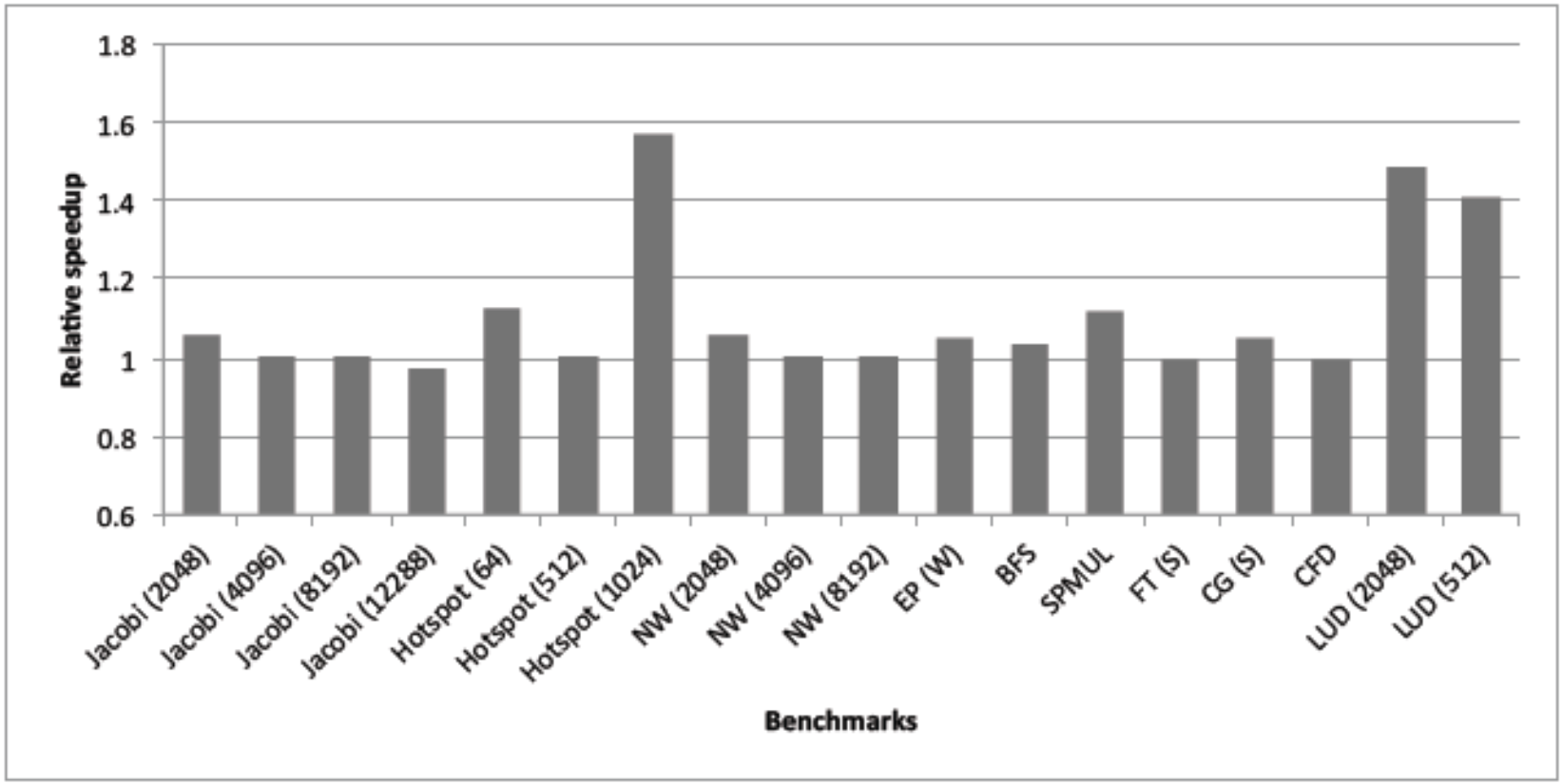
Phase 2
- Computation optimization
- All program configuration environment and code transformation

Phase 3
- Dependent optimization
- useMallocPitch

Phase 4
- Independent optimization
- All data caching optimization

Modified Iterative Elimination (MIE)

- Tuning Results - Program Speedups
  - Relative to PE result



Modified Iterative Elimination (MIE)

- Tuning Results – Tuning time

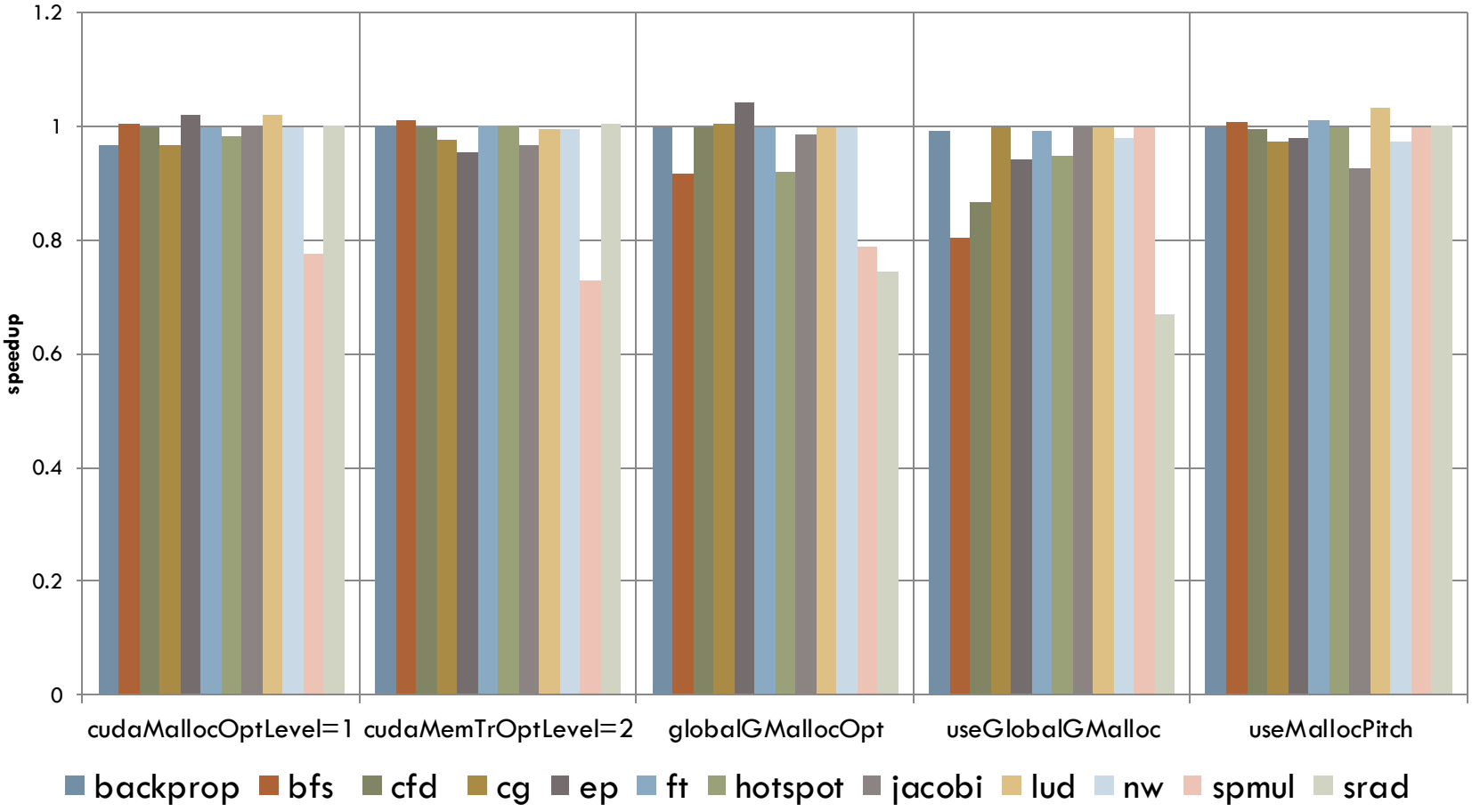
Benchmark	Tuning Time (mins)	
	Pruned Exhaustive Tuning	Modified IE Tuning
SRAD	538	23
FT (S)	2345	23
CG (S)	1108	17
CFD (97k)	1083	210
FT (A)	3680	97
Jacobi (12288)	98	55

IMPACT OF INDIVIDUAL OPTIMIZATION OPTIONS

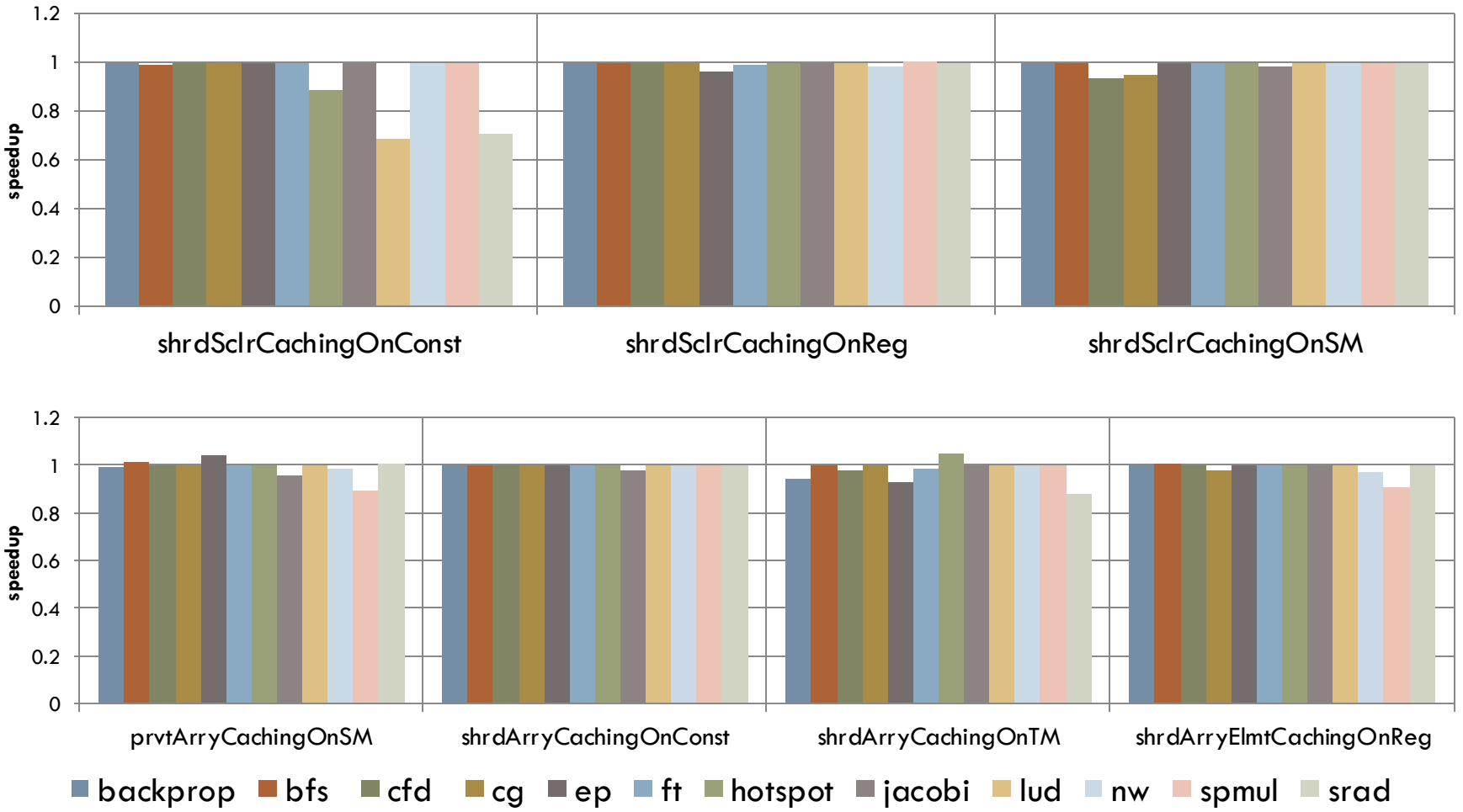
Impact of Individual Optimization Options

- Experiment Setup
  - 3 GHz AMD dual-core processor
  - 12 GB of RAM
  - NVIDIA Quadro FX 5600 GPU
    - 16 SM clocked at 1.35 GHz
    - 1.5 GB of device memory
  - All OpenMPC generated codes are compiled with nvcc with –O3 option

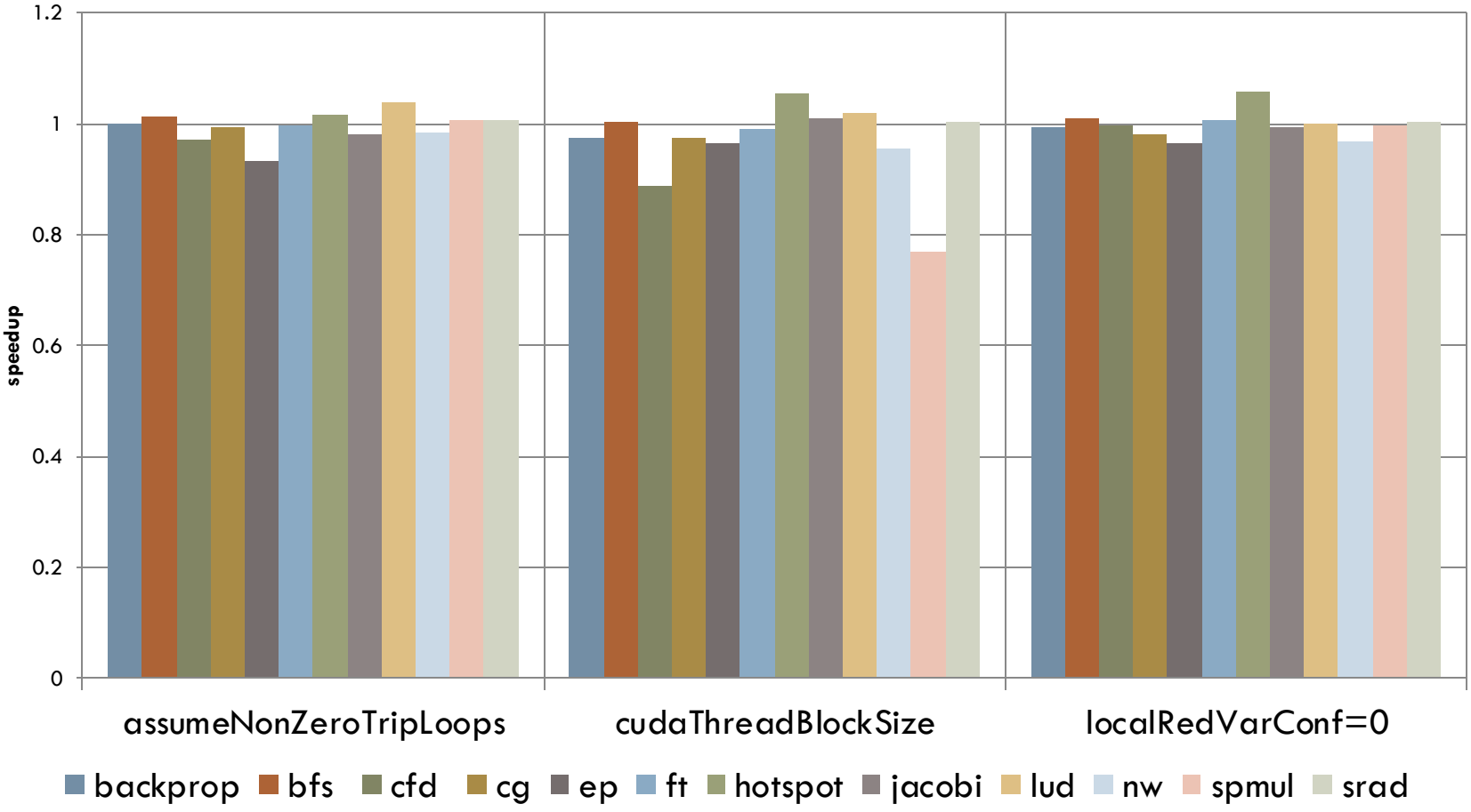
Result-Data Offloading Optimizations



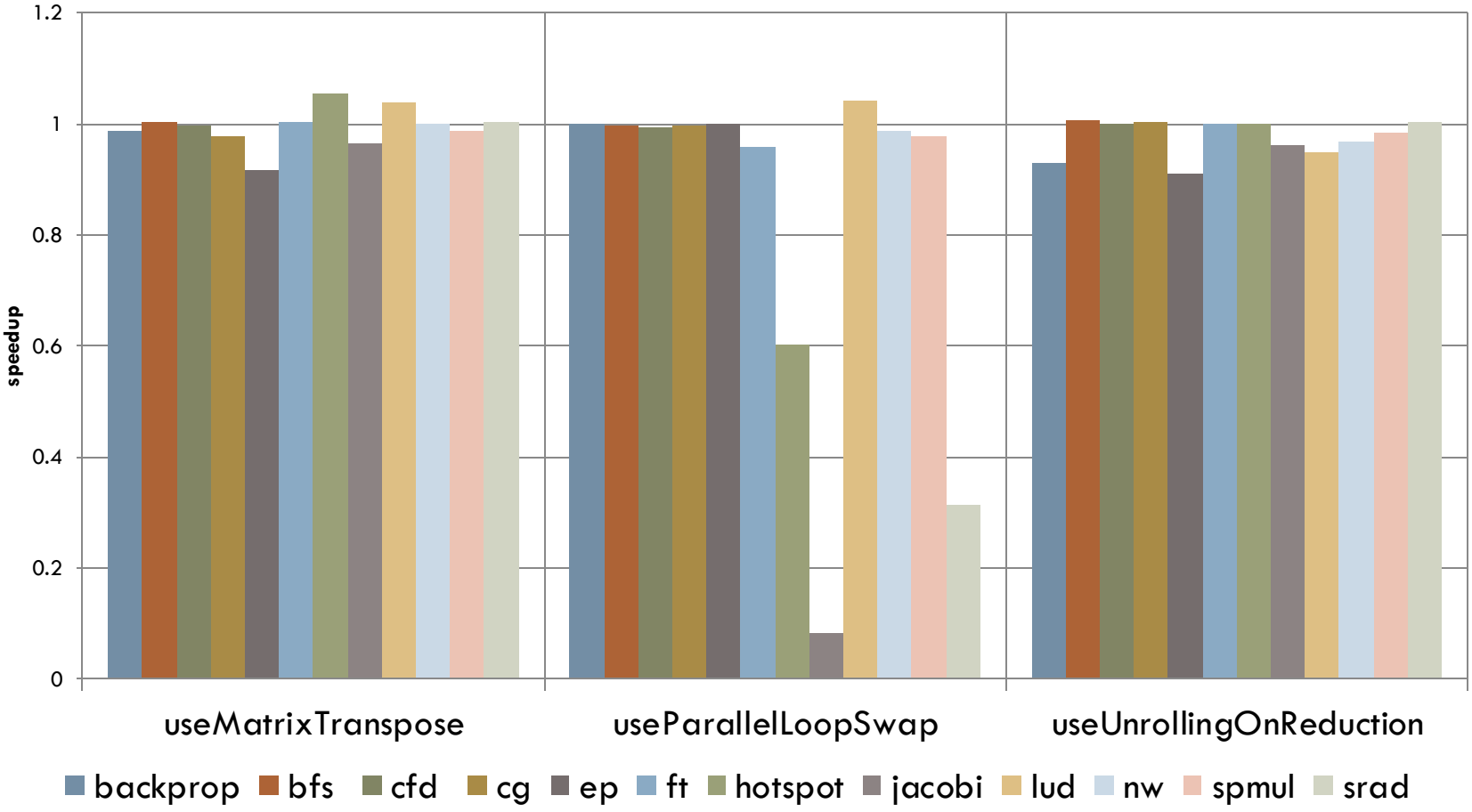
Result-Data Caching Strategy



Result-Program Environment Configuration



Result-Code Transformation



CONCLUSION

Conclusion

- Analyzes the performance of GPU optimization techniques presented in OpenMPC
  - Data-offloading, code transformation, and data caching are important optimizations for GPU programs
  - Explicit GPU programming needs CUDA-extension to specify data-offloading and data-caching for best performance
  - Data-offloading and code transformation optimizations presented in this work should be applicable for any future accelerators
- New Tuning System called MIE
  - Significantly reduce tuning time
  - Be able to tolerate runtime variation

References

- [1] Blume, W., Eigenmann, R.: Performance analysis of parallelizing compilers on the perfect benchmarks programs. IEEE Transactions on Parallel and Distributed Systems 3 (1992) 643-656
- [2] Pan, Z., Eigenmann, R.: Fast and eective orchestration of compiler optimizations for automatic performance tuning. In: Proceedings of the International Symposium on Code Generation and Optimization. CGO '06, Washington, DC, USA, IEEE Computer Society (2006) 319-332