

Optimization Opportunities Created by Global Data Reordering

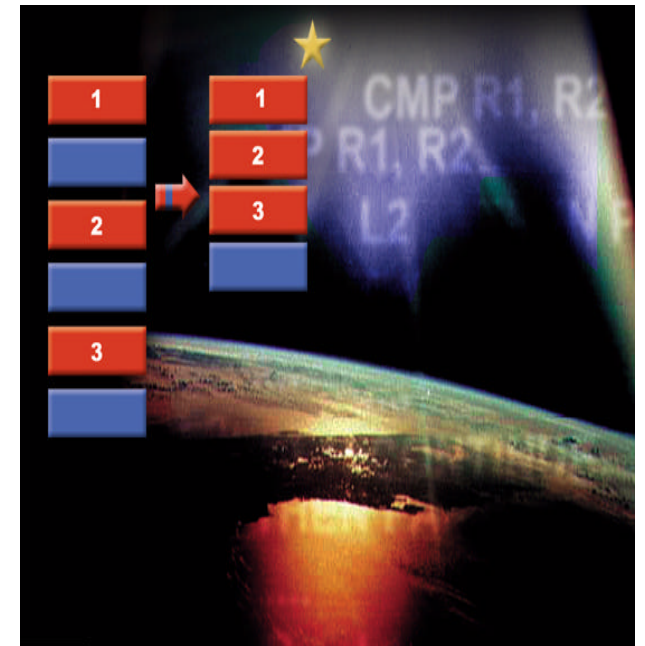
Gad Haber

Moshe Klausner

Bilha Mendelson

Vadim Eisenberg

Maxim Gurevich



IBM Haifa Research Lab

Prior Art and Motivation

■ Prior Art on Data Reordering

- B. Calder, C. Krintz, S. John, and T. Austin, "Cache-Conscious Data Placement", ASPLOS 98.
- R. J. Blainey, C. M. Donawa, and McInnes L., "Global Variable Coalescing", U.S. Patent No 5,850,549, IBM

■ Prior Art on Optimizing Global Data Accesses

- Srivastava A., D. W. Wall, "Link-Time Optimizations of Address Calculation on a 64-bit Architecture", Research Report 94/1, Digital.
- Muchnick S. S., "Advanced Compiler Design & Implementation"

■ In this work we try to combine the two



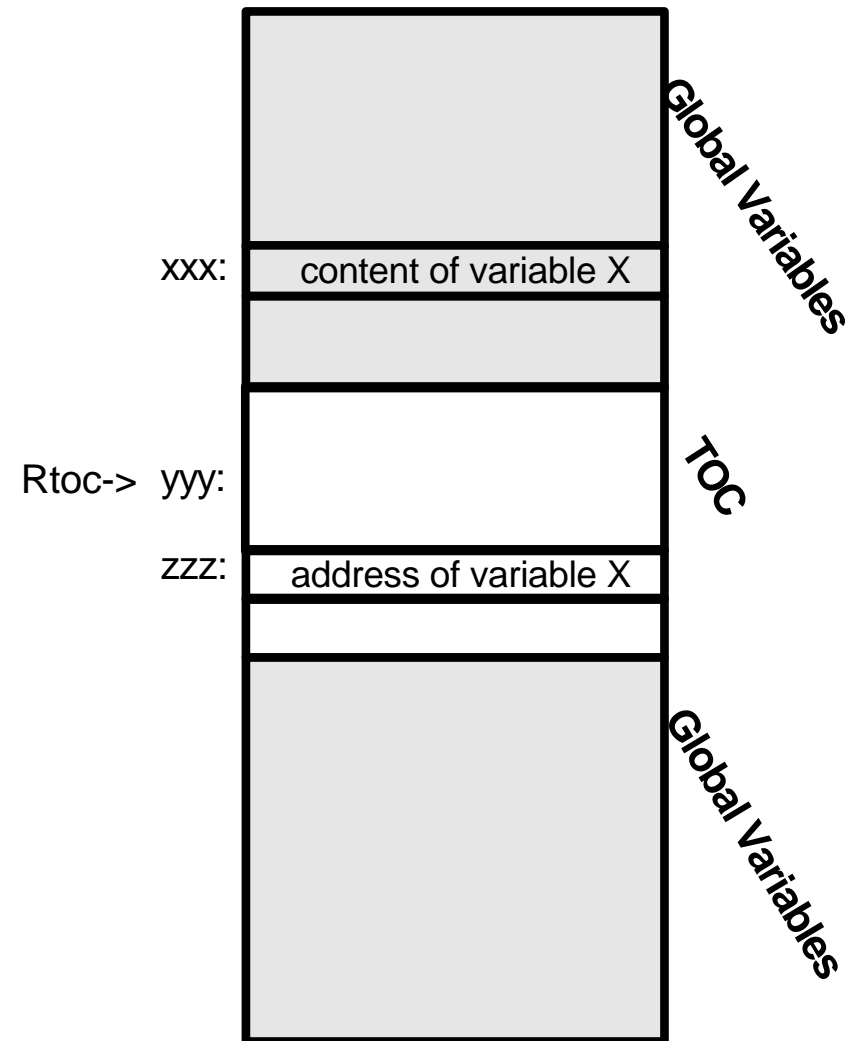
Global Data Mechanism in RISC Architectures

- **Loading X addr from TOC:**

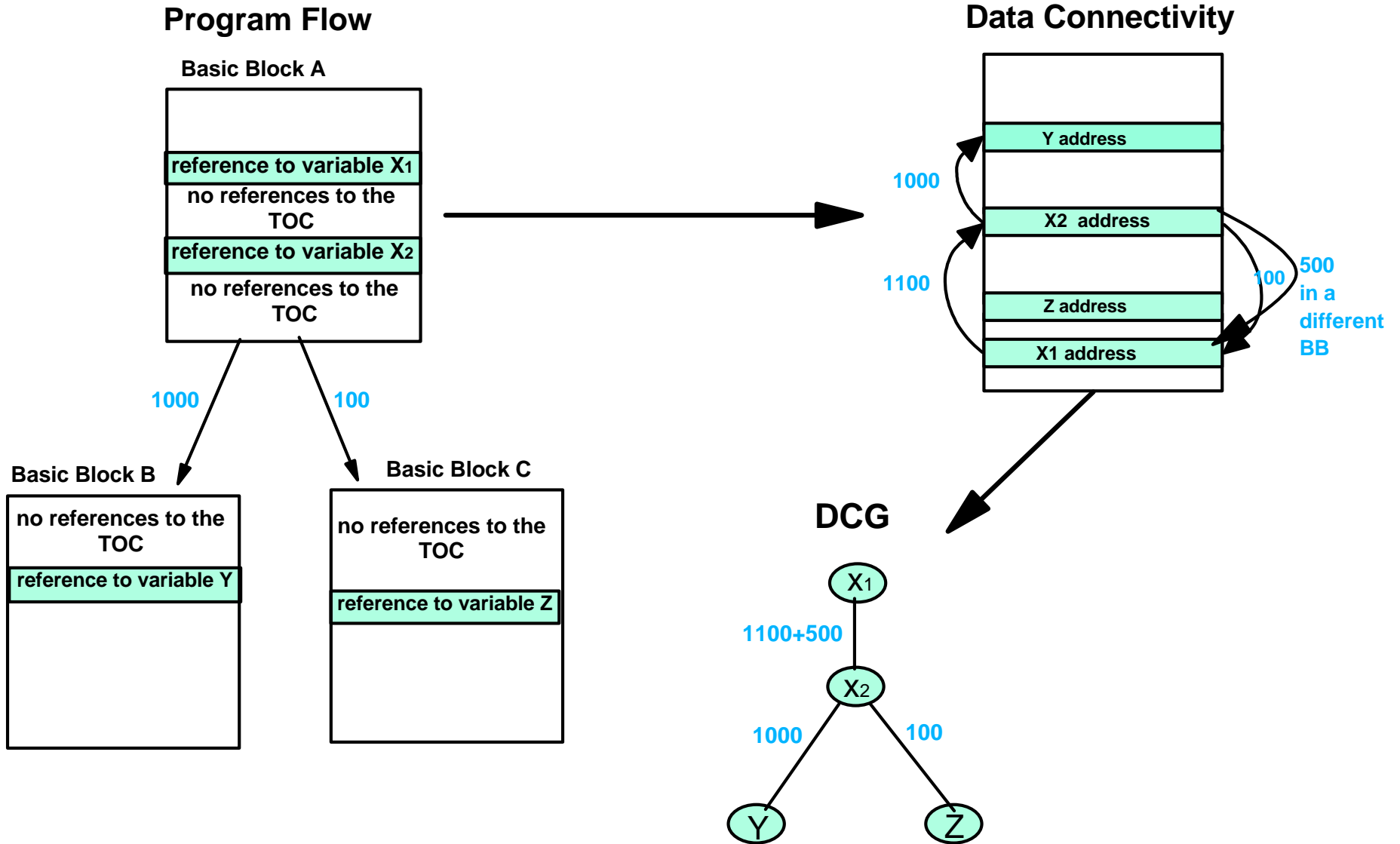
```
load R5,disp(Rtoc)
disp = zzz-yyy
```

- **Calculating X addr:**

```
addi R5,Rtoc,imm1
addis R5,R5,imm2
imm1 = LSB(zzz-yyy)
imm2 = MSB(zzz-yyy)
```



Data Connectivity Graph



Global Data Optimization Algorithm

1. Create the Data Connectivity Graph
2. Relocate constants from the code to the Global Data area
3. Relocate the TOC to the beginning of the global data area
4. Place all global variables using the Data Placement Algorithm
5. Reorder TOC entries in the order of corresponding global variables
6. Mark all the TOC entries that can be removed
7. Until there are no more removable TOC entries do:
 - ▶ Remove all TOC entries that are marked removable
 - ▶ Update the TOC Anchor to point to $\text{TOC}_{\text{start}} + \text{RANGE}_{\text{load}}/2$.
 - ▶ Mark as removable, TOC entries of vars in the range of TOC anchor
 - ▶ Update references to global variables



Setting Node's Hotness in the DCG

- $INSTR_v$ - set of instructions that reference TOC entry v of global variable V
- $EXEC(instr)$ - execution count of instruction $instr$
- $H(v)$ - Variable Hotness
 - ▶ $H(v) = \sum_{instr \in INSTR_v} EXEC(instr)$
- $NH(v)$ - Normalized Hotness measure of a variable V
 - ▶ $SIZE(v)$ is the size of the variable V
 - ▶ N is a normalization factor between 0 and 1
 - ▶ $NH(v) = (H(v)/SIZE(v)) * N + H(v) * (1 - N)$



Data Placement Algorithm

- Based on DCG the algorithm constructs chains
 - ▶ Head of new chain: node v with highest $NH(v)$
 - set $CurrTocEntry = v$
 - set $CurrHotness = NH(v)$
 - ▶ Next in chain: Node u with the most heavy weighted edge ($CurrTocEntry, u$) in the DCG from $CurrTocEntry$ to other TOC entry u
 - set $CurrTocEntry = u$
 - ▶ Start new chain when:
 - No available edge or
 - $NH(u) / CurrHotness \geq HT$ (Hotness Threshold)
 HT is a given parameter between 0 and 1



Experimental Results on SPECint2000

■ Platform:

- ▶ IBM 64-bit SMP Power3 machine
- ▶ Two processors
- ▶ 64KB L1 data cache, 32KB instruction cache
- ▶ 4MB unified L2 cache.

■ OS:

- AIX version 4.3.3.

■ Compiler:

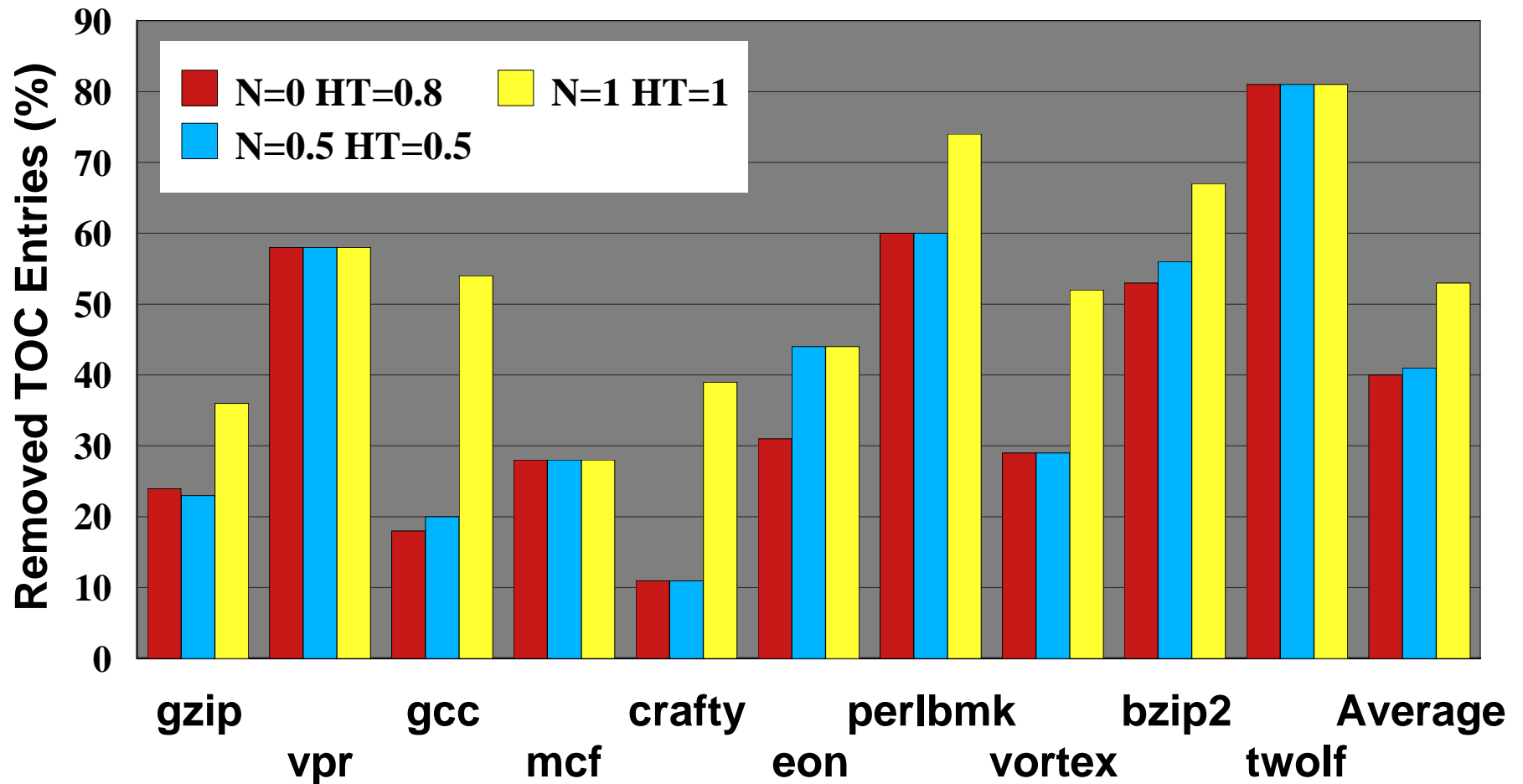
- IBM xLC -O3

■ Optimization:

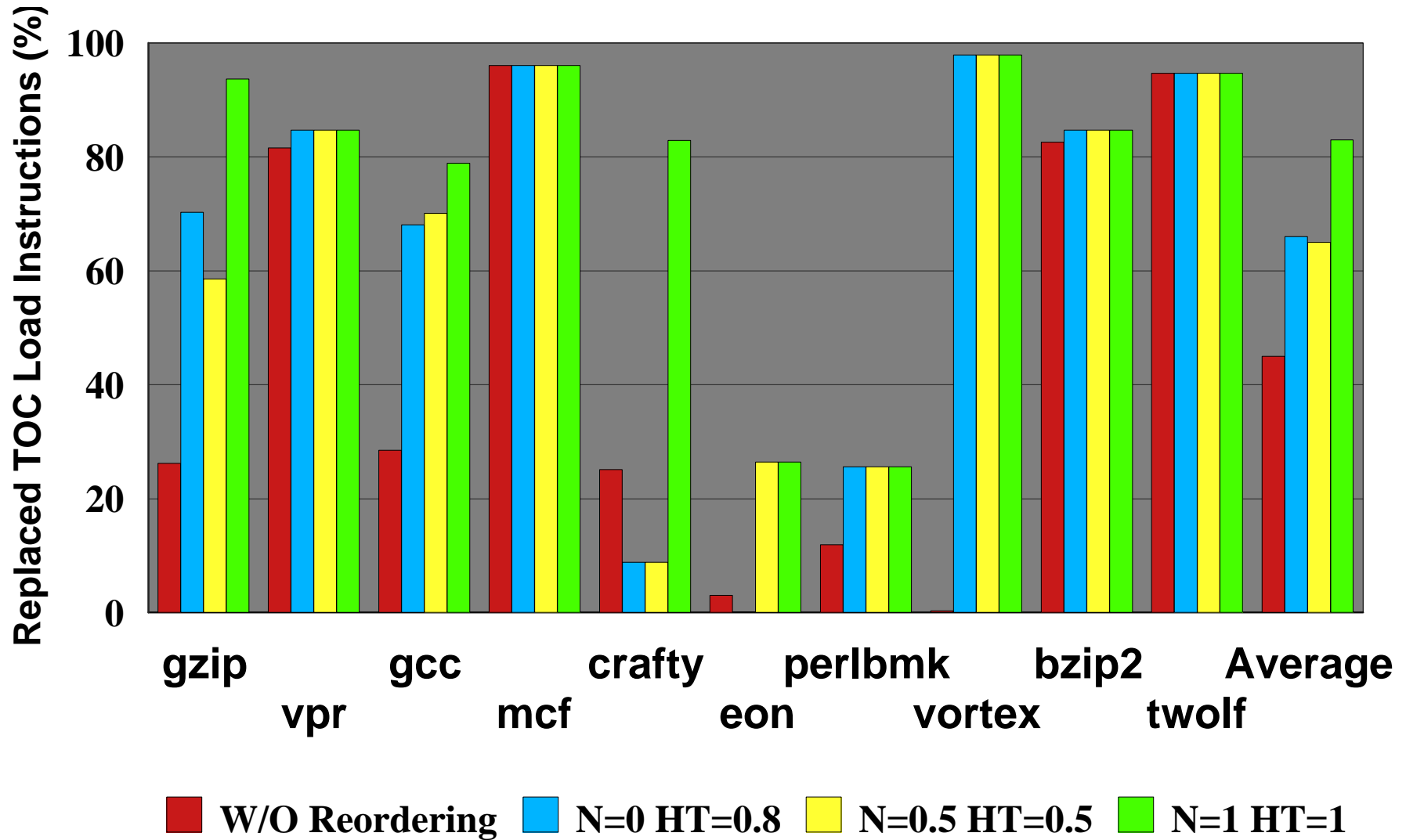
- Implemented into FDPR - IBM post-link optimizer
- Feedback information gathered with the train input.
- Execution time measured on the ref input.



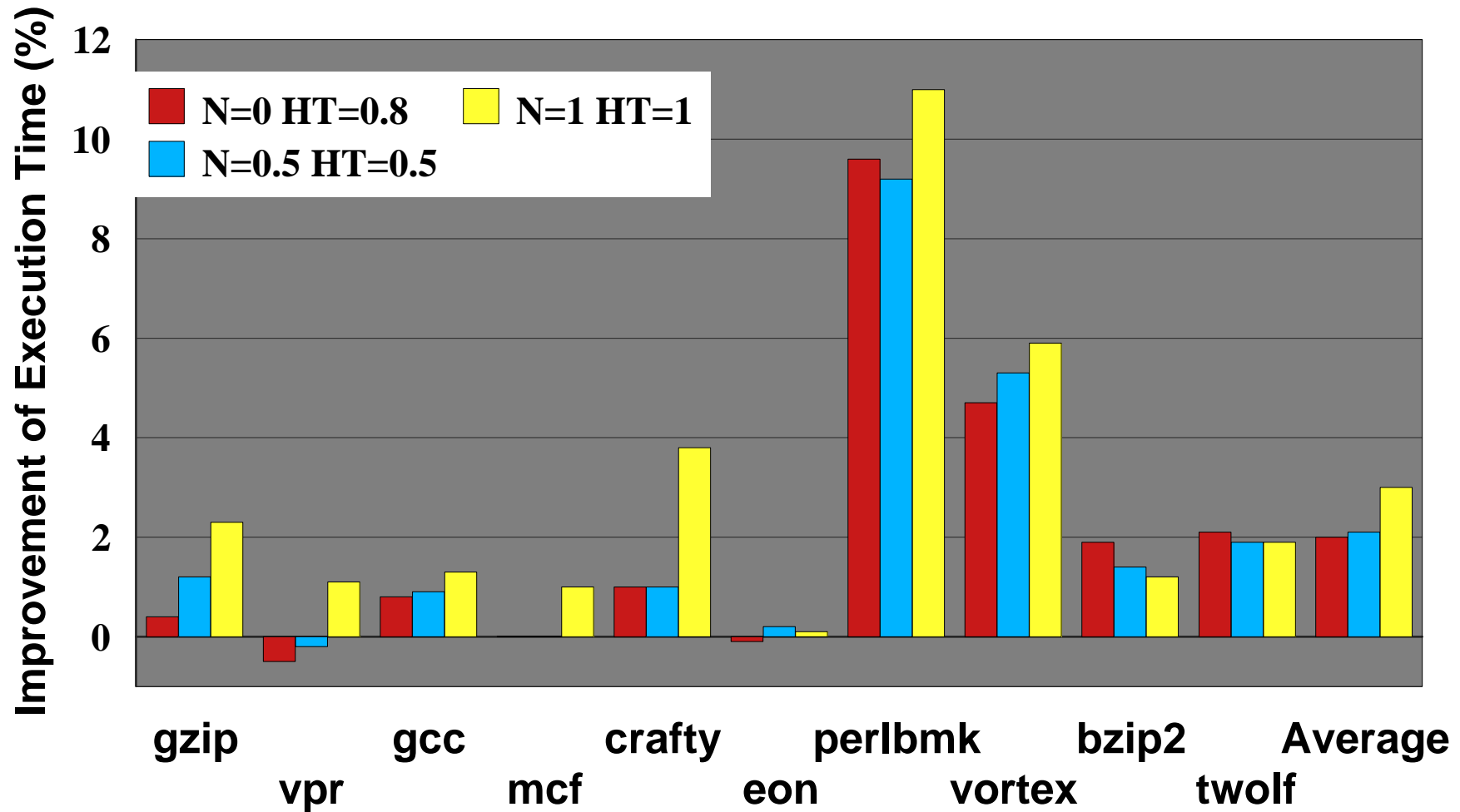
Removed TOC Entries



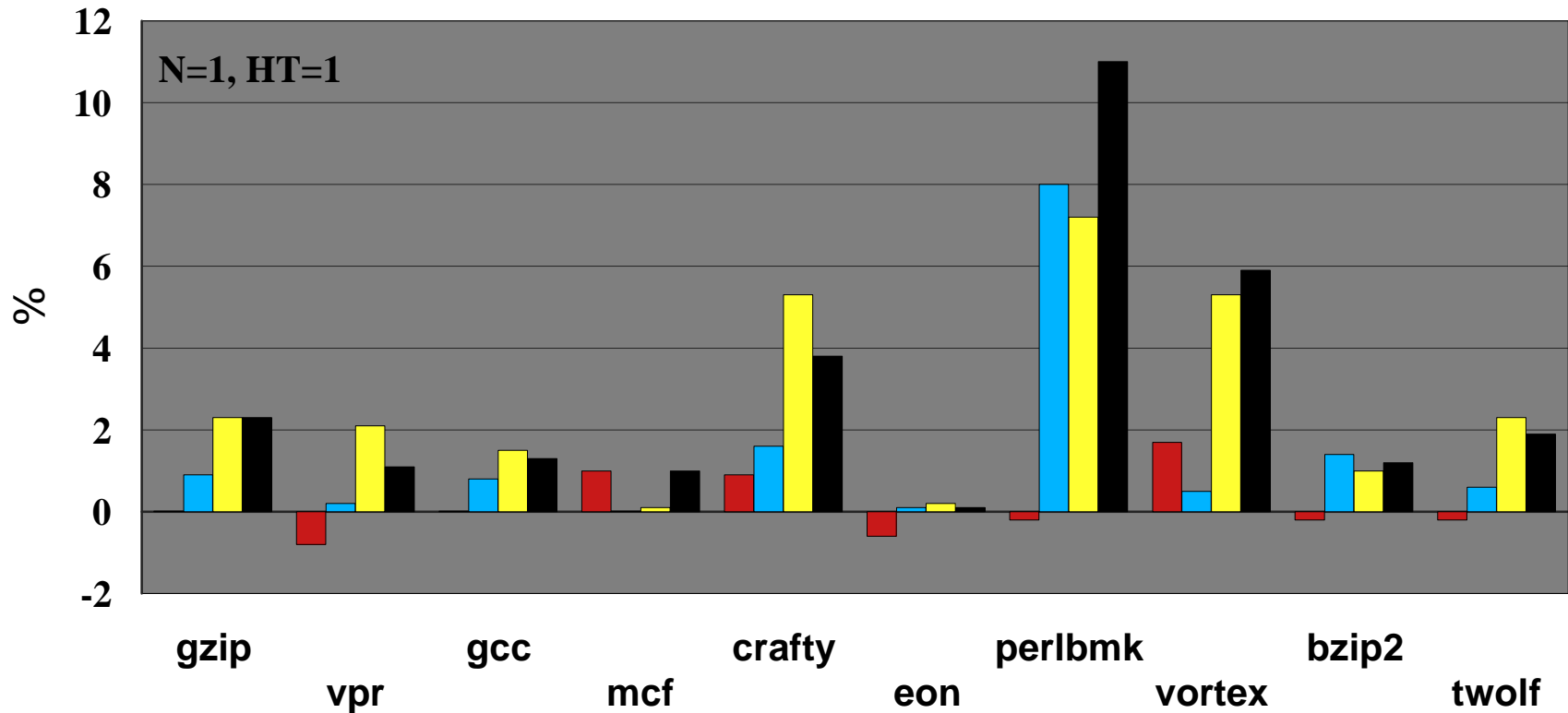
Replaced TOC Load Instructions



Execution Time Improvement



Performance Analysis



- Execution time improvement (%) using data reordering alone
- Execution time improvement (%) using TOC optimization w/o data reordering
- Replaced Load Executions/Total instructions Execution (%)
- Execution time improvement (%) using TOC optimization + data reordering



Conclusions

- Improving performance by data reordering alone, is not enough
- In order to obtain significant performance gain
 - Additional optimization opportunities resulting from reordering techniques need to be exploited
 - For example:
 - ▶ Reducing branch penalty effects during code reordering
 - ▶ Reducing fragmentation during heap reallocation
 - ▶ Reducing the number of Load instructions by using global data reordering

