Frequently Used Trace Cache

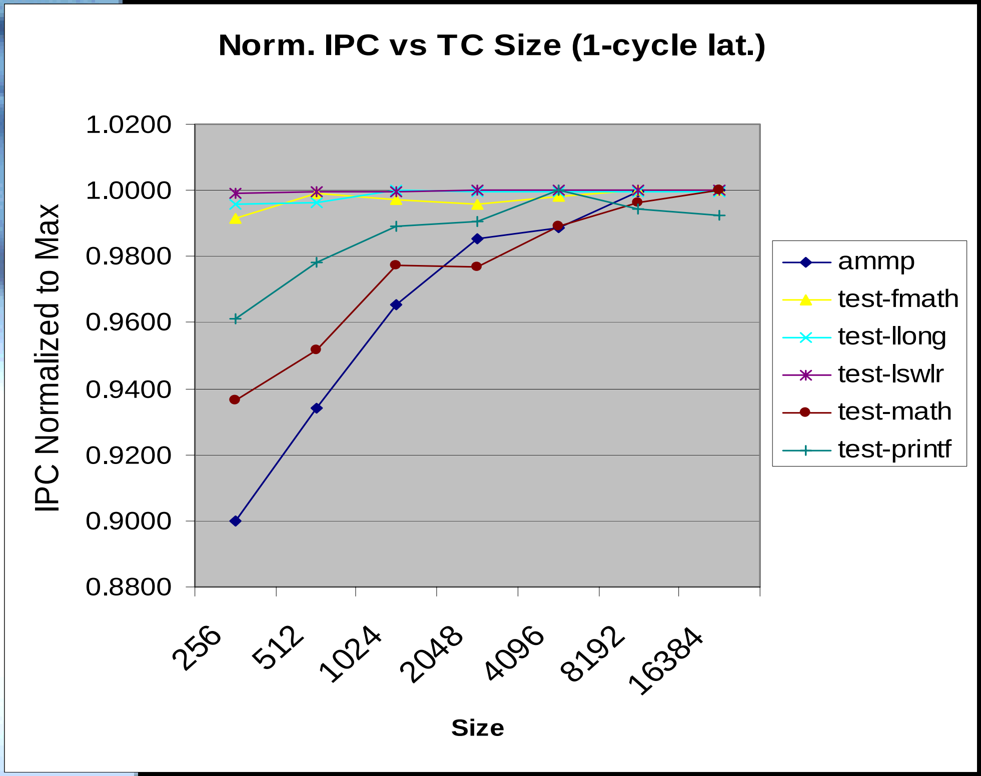
```
[efurbish@hoss simplesim-2.0]> wc -l trace_cache.c
     666
```

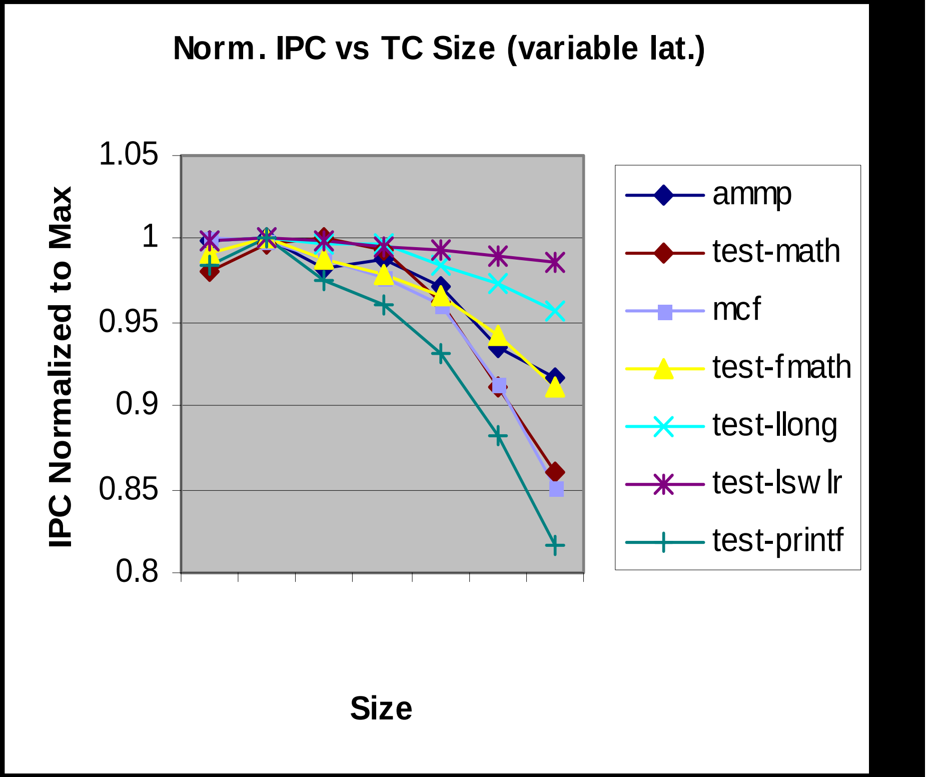
Advanced Computer Architecture,
under Dr. Scott Rixner

Marc Power
Eric Furbish
Indraneel Datta

M

- Large trace caches suffer increased latency
  - Lowers performance below ideal





- We’d like to minimize the performance lost to latency.

Co

- To recover from the ill effects of latency in the trace cache, we need some *low latency* means of getting at the same data.
- Normally, solve by implementing L1/L2 cache hierarchy
  - Contention in the L1 can evict frequently used lines with infrequently used ones
  - Contention could possibly be reduced by intelligently filling the L1
- *Frequently Used Trace Cache (FUTC)* –
  - Single-cycle latency, small L1
  - Judiciously filled from L2 with frequently used lines
- Given the new gizmo, we hoped:
  - $IPC(TC_s + FUTC_i) > IPC(TC_s + L1TC_i) > IPC(TC_s)$

FU

- One saturating counter per trace cache line
  - Incremented on read hit
- Counters cleared on TC writes
  - Rewrites not propagated into FUTC
    - Saves read port
    - Maintains logical meaning of counters (apply to a *specific* trace)
- Lines with counters over threshold are promoted
  - Promoted on a read hit
  - Counter is not reset

FU

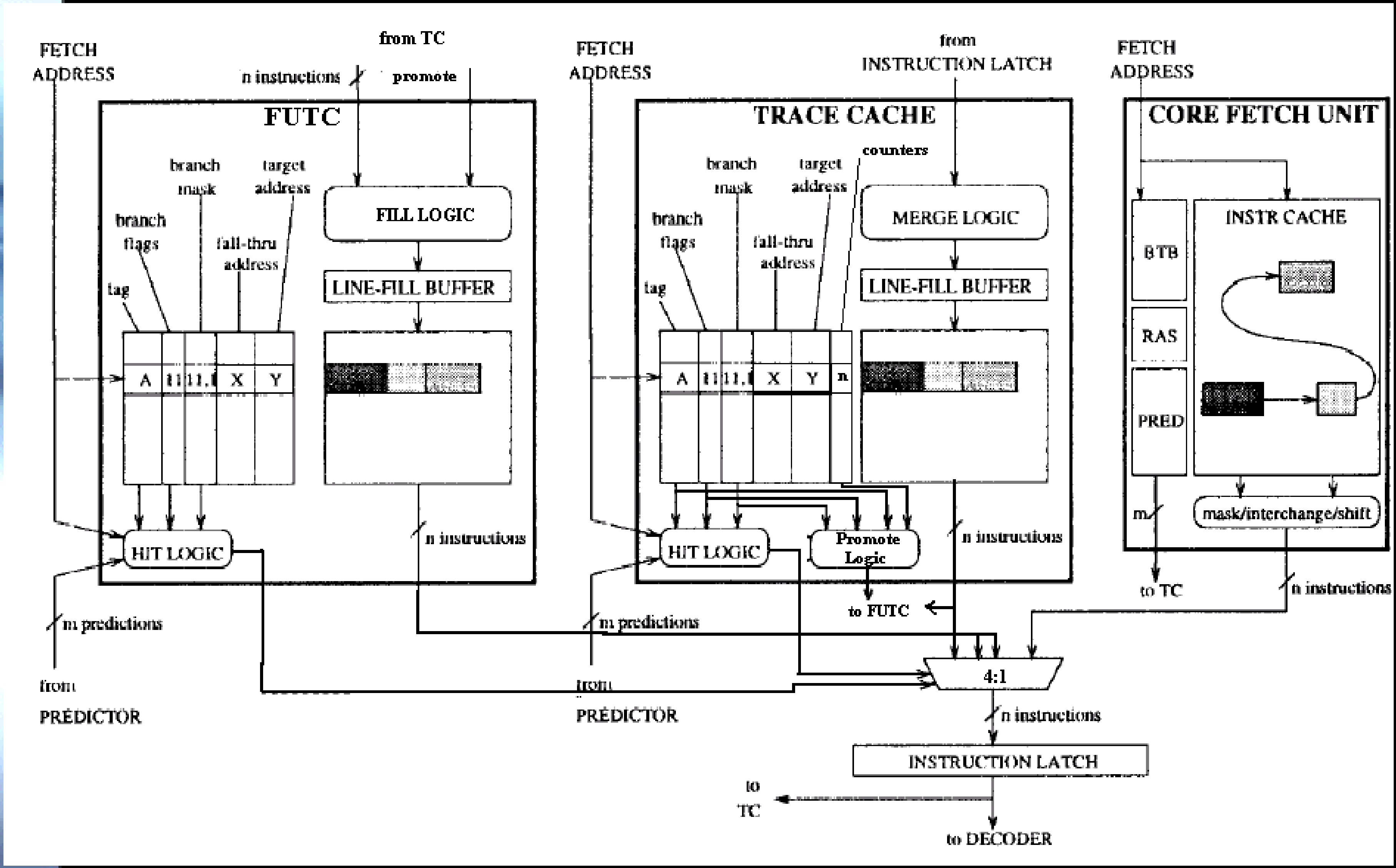


Figure Modified from (E. Rotenbert, S. Bennett, J. Smith. Trace cache: a low latency approach to high bandwidth instruction fetching. Tech Report 1310, CS Dept., Univ. of Wisc.-Madison, 1996)

Ar

- Size of FUTC (8-64KB is reasonable) in order to remain under single-cycle latency
- Counters consume 1-3KB for 1MB trace cache
  - Assumes threshold of 1-8 (1-3 bits)
- Power cost is minimal
  - No worse than L1 trace cache
  - Single-ported for speed and power

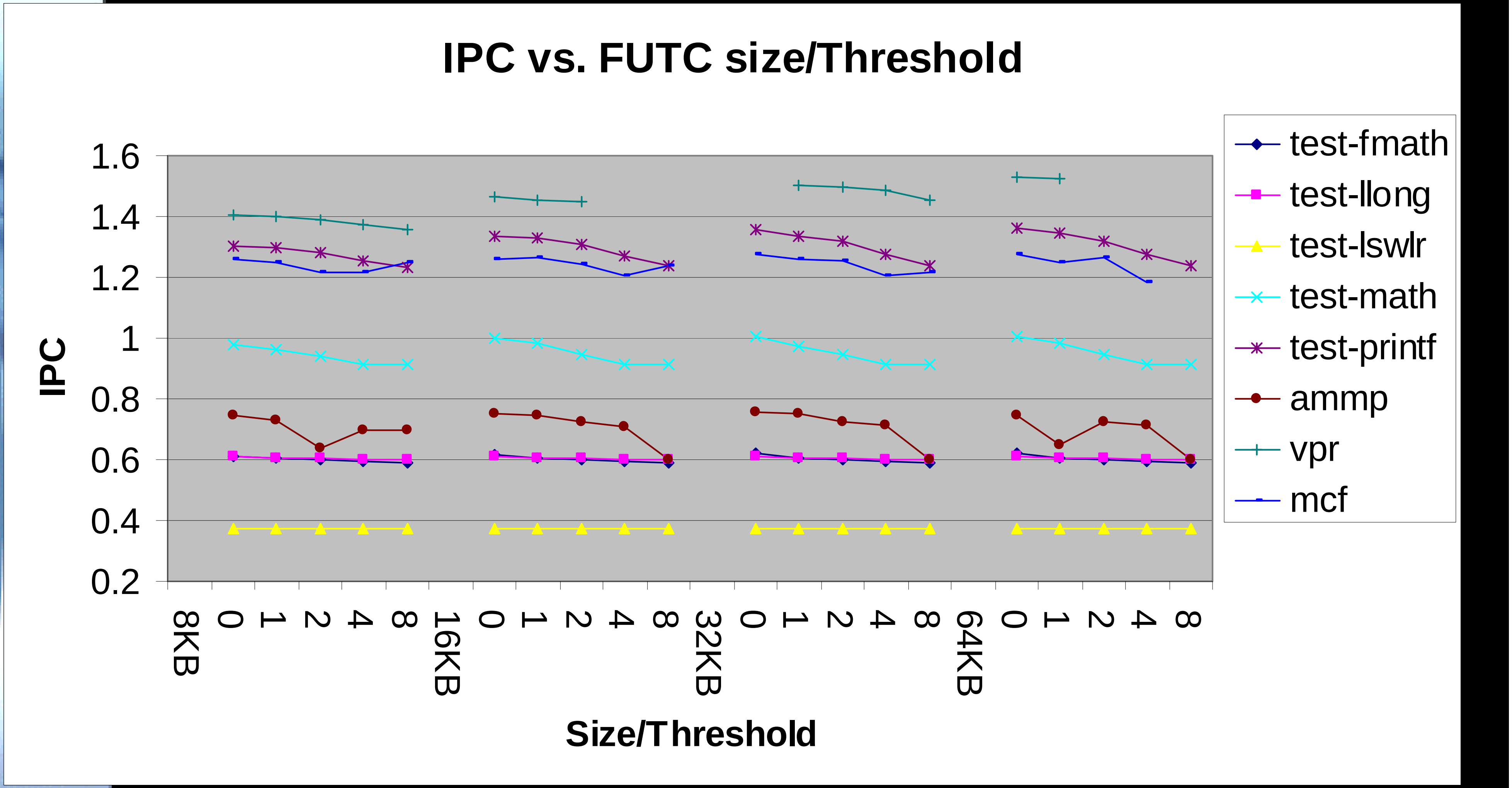
M

1. Use “infinite” backend to stress fetch mechanism
2. Select ‘Optimal’ TC Size
   - Based on ideal (single-cycle) performance vs. performance with latency accounted for
   - Pick best candidate for improvement with FUTC
3. Run varying “approximate L1” sizes (8-64KB)
   - L1 = FUTC with threshold 0
4. Run varying FUTC sizes (8-64KB) with thresholds of 1, 2, 4 and 8
5. Compare TC/FUTC and L1/FUTC

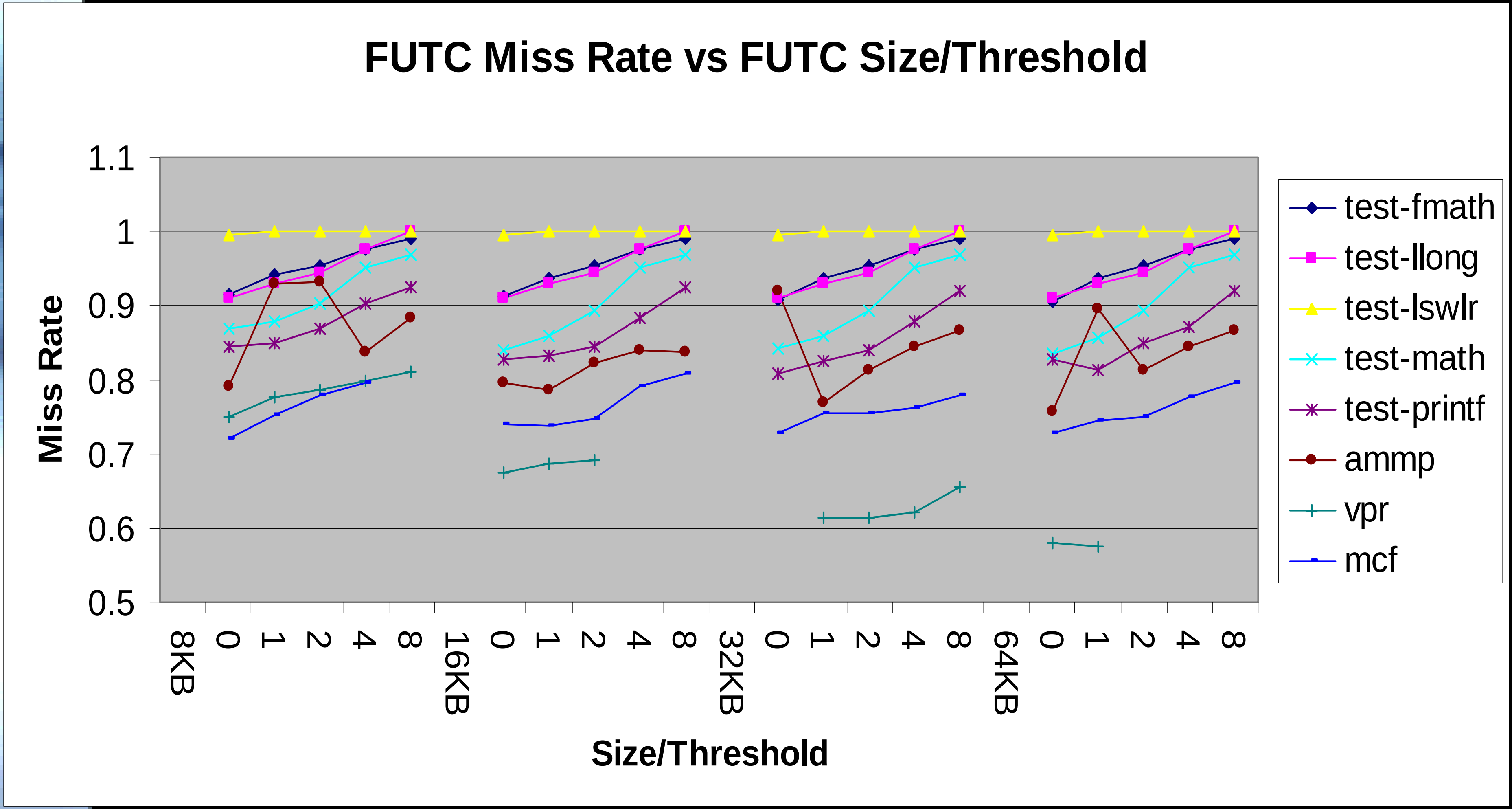
V

- Branch predictor accuracies of 90-99%
  - Exactly the range we would expect
- Performance and trace cache hit rates very similar to trace cache paper
- Dispatch and commit streams are identical to unmodified sim-outorder
  - Program flow is guaranteed to be correct
- It has seen billions of instructions without flaws

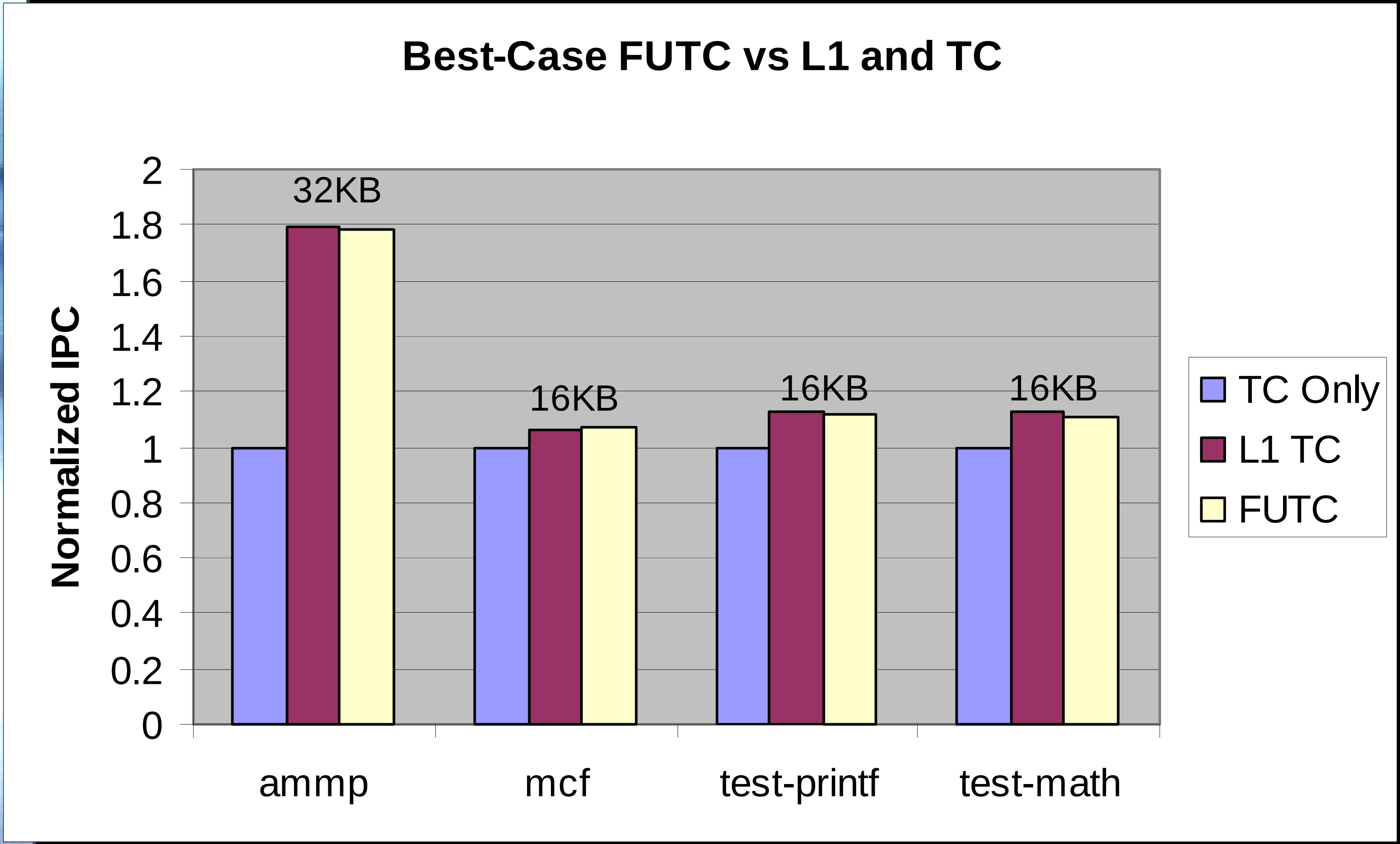
Re


IPC vs. FUTC size/Threshold
IPC
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
8KB
0
1
2
4
8
16KB
0
1
2
4
8
32KB
0
1
2
4
8
64KB
0
1
2
4
8
Size/Threshold
test-fmath
test-llong
test-lswlr
test-math
test-printf
ammp
vpr
mcf

Re


FUTC Miss Rate vs FUTC Size/Threshold
Miss Rate
1.1
1
0.9
0.8
0.7
0.6
0.5
8KB
0
1
2
4
8
16KB
0
1
2
4
8
32KB
0
1
2
4
8
64KB
0
1
2
4
8
Size/Threshold
test-fmath
test-llong
test-lswlr
test-math
test-printf
ammp
vpr
mcf

Be


Best-Case FUTC vs L1 and TC
Normalized IPC
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
32KB
16KB
16KB
16KB
ammp
mcf
test-printf
test-math
TC Only
L1 TC
FUTC

Fu

- Run larger programs
  - Increased contention could make FUTC effective
- Use partial matching and inactive issue
  - Higher TC hit rate = more promotions = more contention

Co

- Hypothesis was partly correct
  - Small auxiliary cache always helps over TC
- Contention in L1 outweighed by FUTC “warmup”
- New Hypothesis:
  - $IPC(TC_s + L1\ TC_i) > IPC(TC_s + FUTC_i) > IPC(TC_s)$
  - Really: $IPC(TC_s + AUX\text{-}TC_i) > IPC(TC_s)$