Power Reduction in VLIW Processor with Compiler Driven Bypass Network

Neeraj Goel

Dr Anshul Kumar

Dr Preeti Ranjan Panda

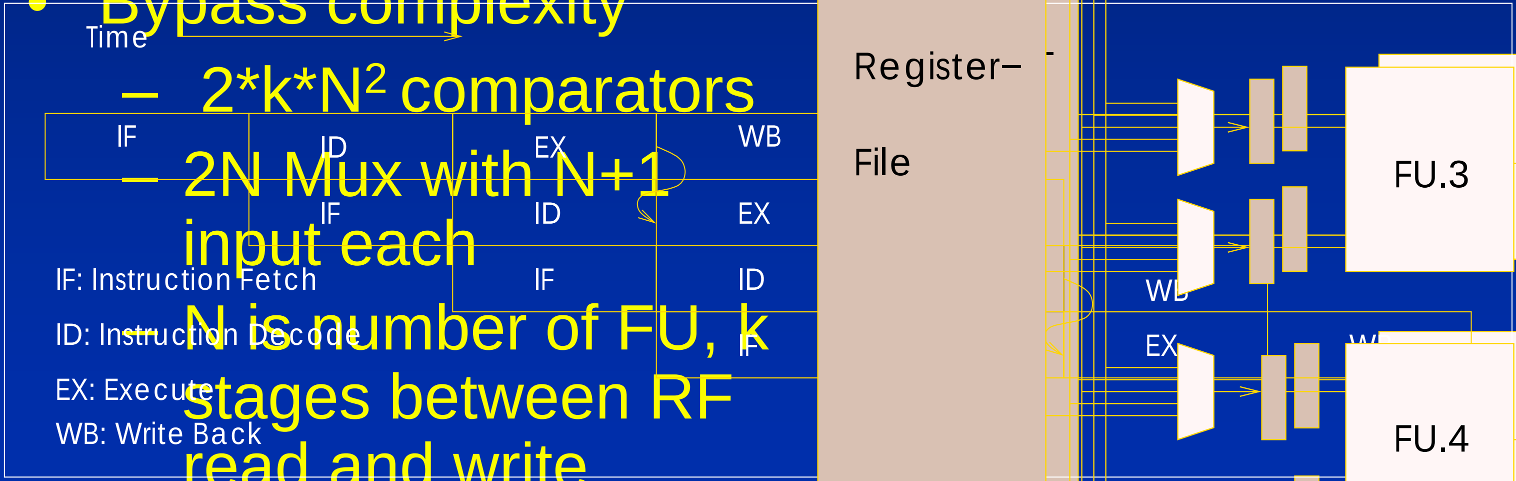
Department of Computer Science, IIT Delhi

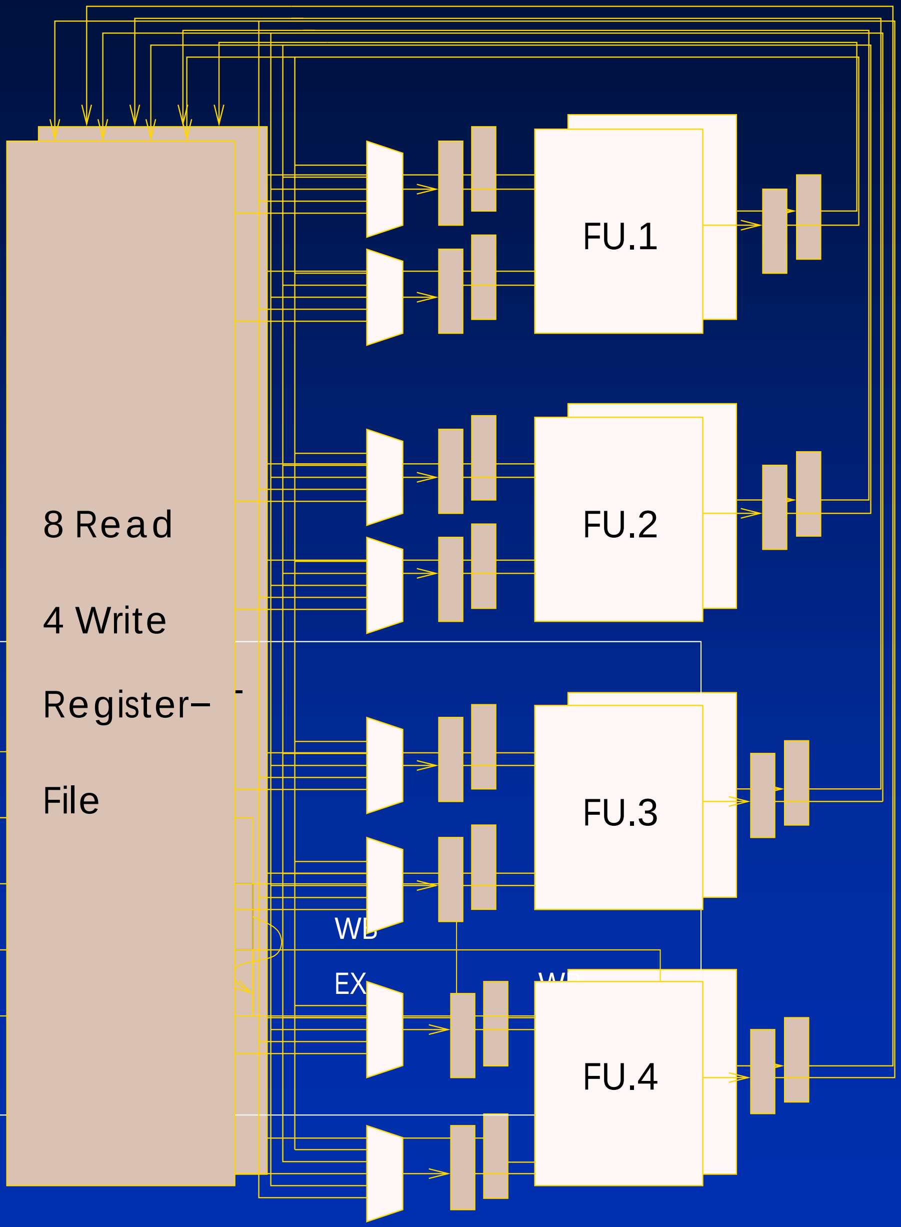
Outline

- Introduction: VLIW, bypass
- Approach
- Techniques
  - Compiler support
  - Micro-architectural modifications
- Experiments
- Conclusion

Introduction

- VLIW Processor
  - Multiple FUs
  - Static scheduled
  - Multi-ported RF
- Bypass
- Bypass complexity
  - $2*k*N^2$ comparators
  - 2N Mux with N+1 input each
  - N is number of FU, k stages between RF read and write





Motivation

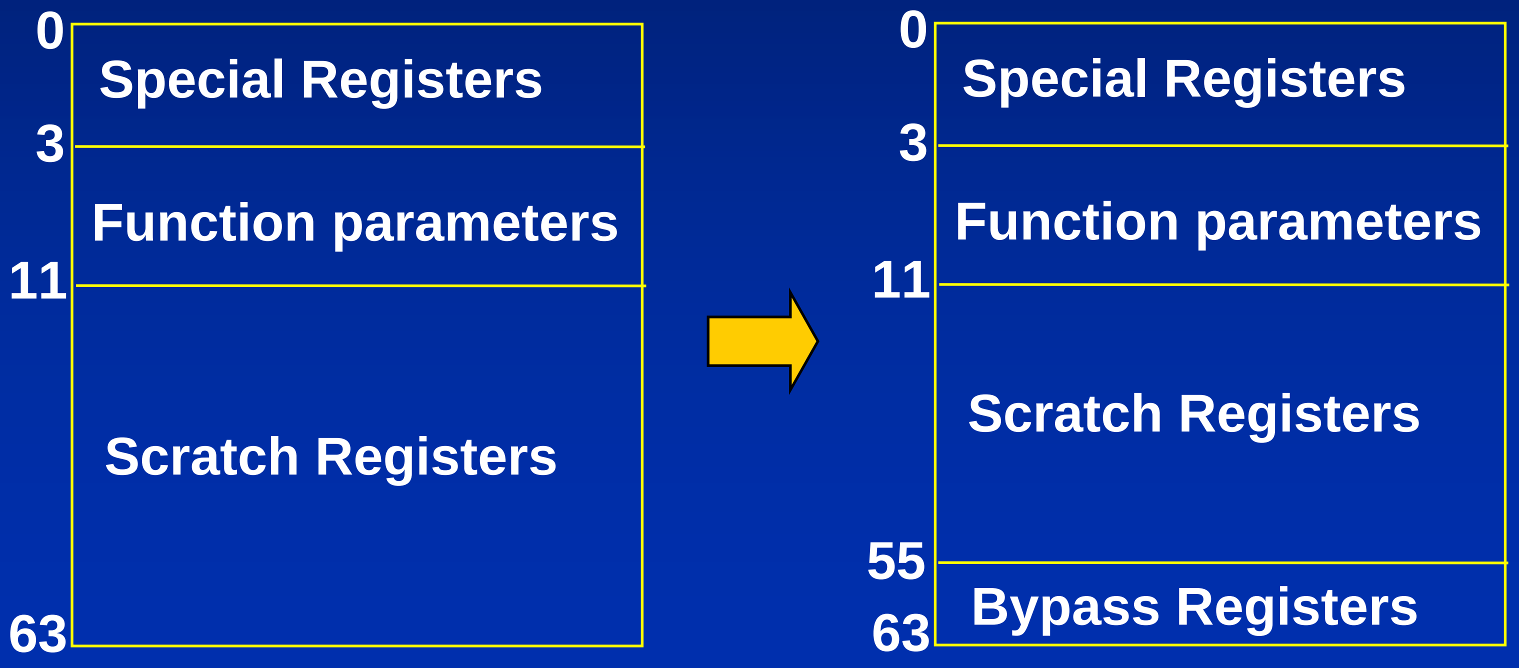
- Conventional pipeline and register file read
  - Read in 2nd stage and bypass in 3rd stage or later
- When reading from bypass, RF read is not necessary
- Most values are short lived
  - Majority of operands are read from bypass network
- RF read is costlier in multi-port RF
  - Read energy increases as N^3, N is no. of ports

Avoidance of redundant RF reads

- Hardware
  - Bypass control need to be before RF read
    - Increase in clock cycle or
    - Increase in number of pipeline stages
- Software
  - Static scheduling in VLIW processor → accurate bypass calculations in compiler
  - Bypass Hints with each operand => increase in instruction width

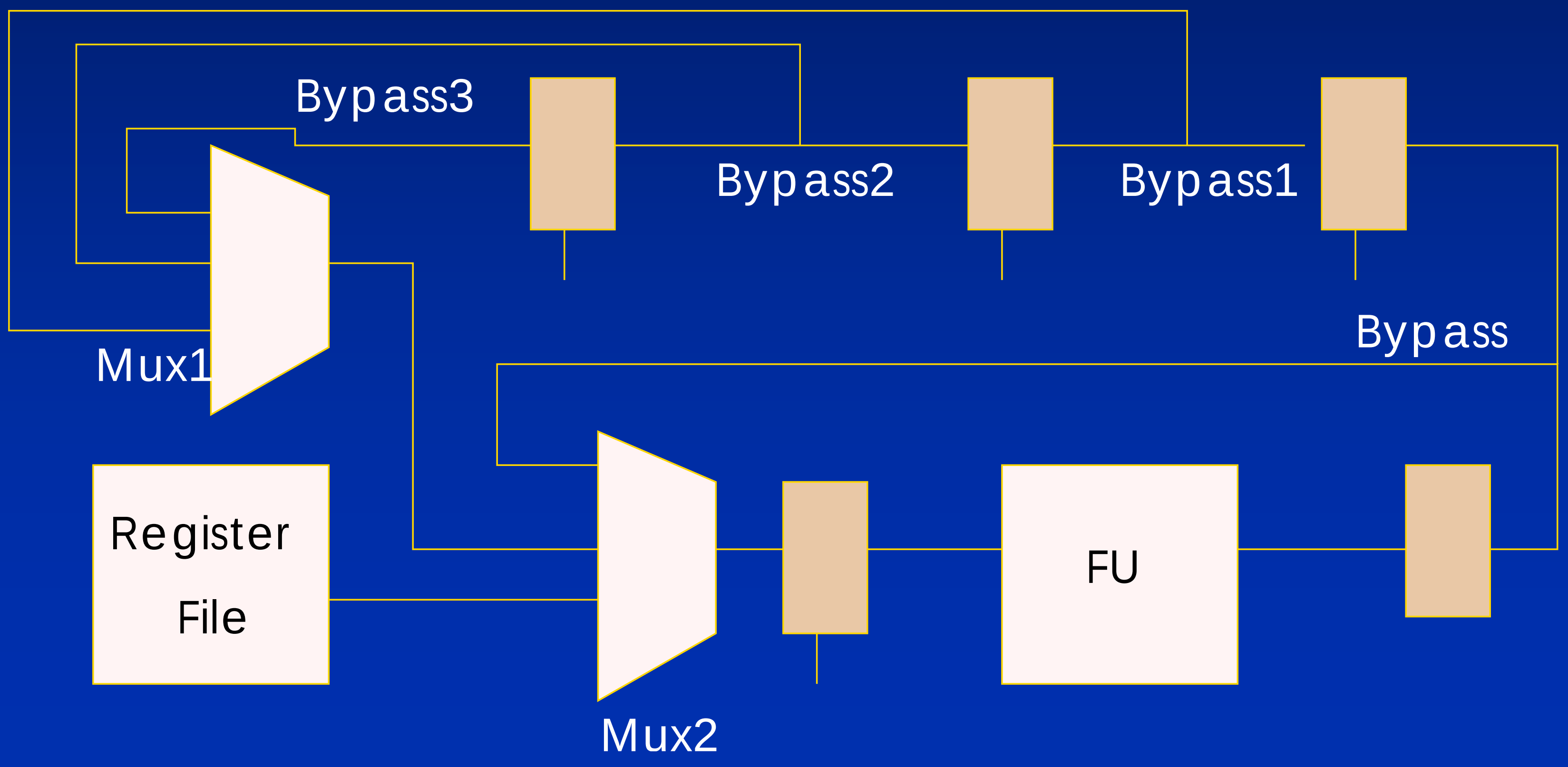
Bypass Control Representation

- Compiler visible bypass registers
- Addresses are assigned to bypass registers
- An example organization



Multi-stage Bypass

- Result is kept in pipeline for more stages
- Hierarchy of multiplexer for performance
- More energy efficient



Avoidance of Redundant Writes

- Values read from bypass are written back to RF as
  - May be required in future
- Values read from bypass, need not to written to RF if:
  - They are not required in future
- Most registers have short life time => more advantageous

Compiler Support

- Analysis
  - Liveness analysis to find if an operand is a “last use” or not
- Code generation
  - Reduce available registers in processor description
  - Register renaming

```
add r13, r14, r8
sub r6,  r13, r9
```

```
add r13, r14, r8
add by1, r14, r8
sub r6,  by1, r9
```

R13 is “last use” in *sub* instruction

R13 is not “last use” in sub instruction

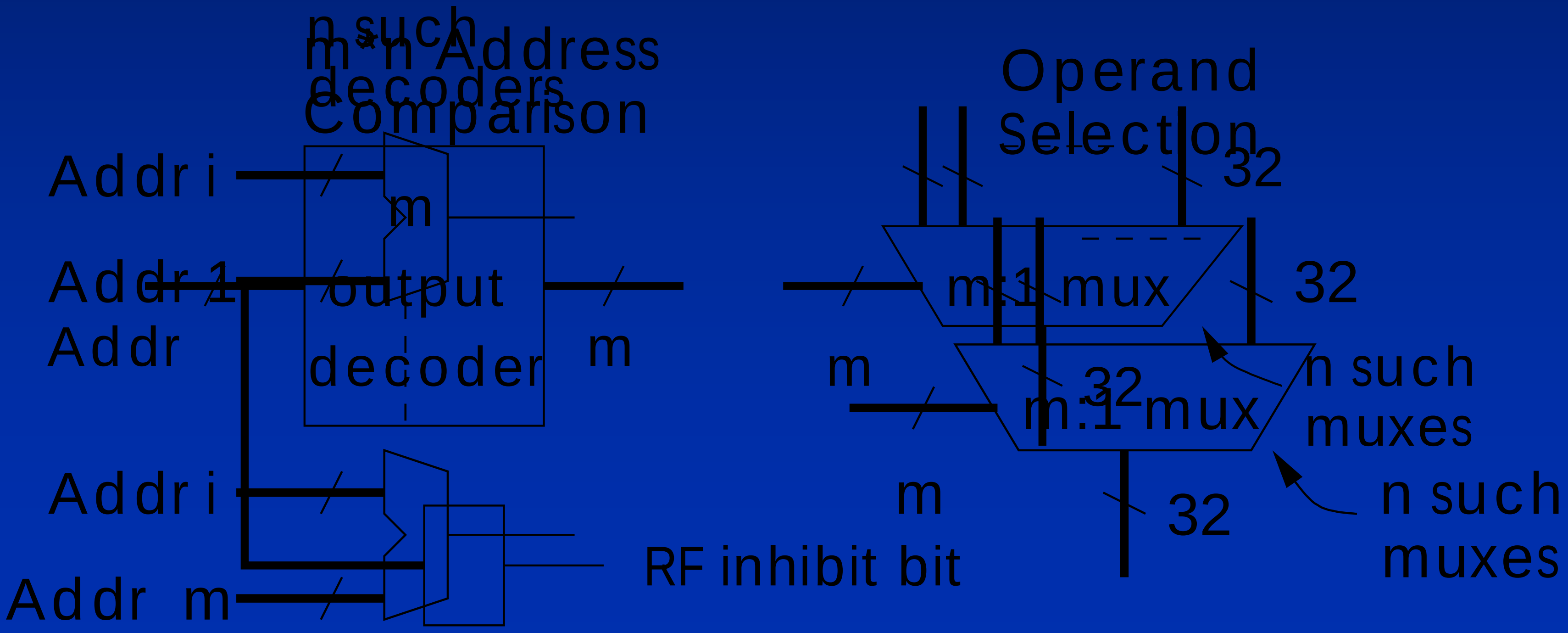
Micro-Architectural Modification

Bypass control in trivial approach

Bypass control in compiler aware approach

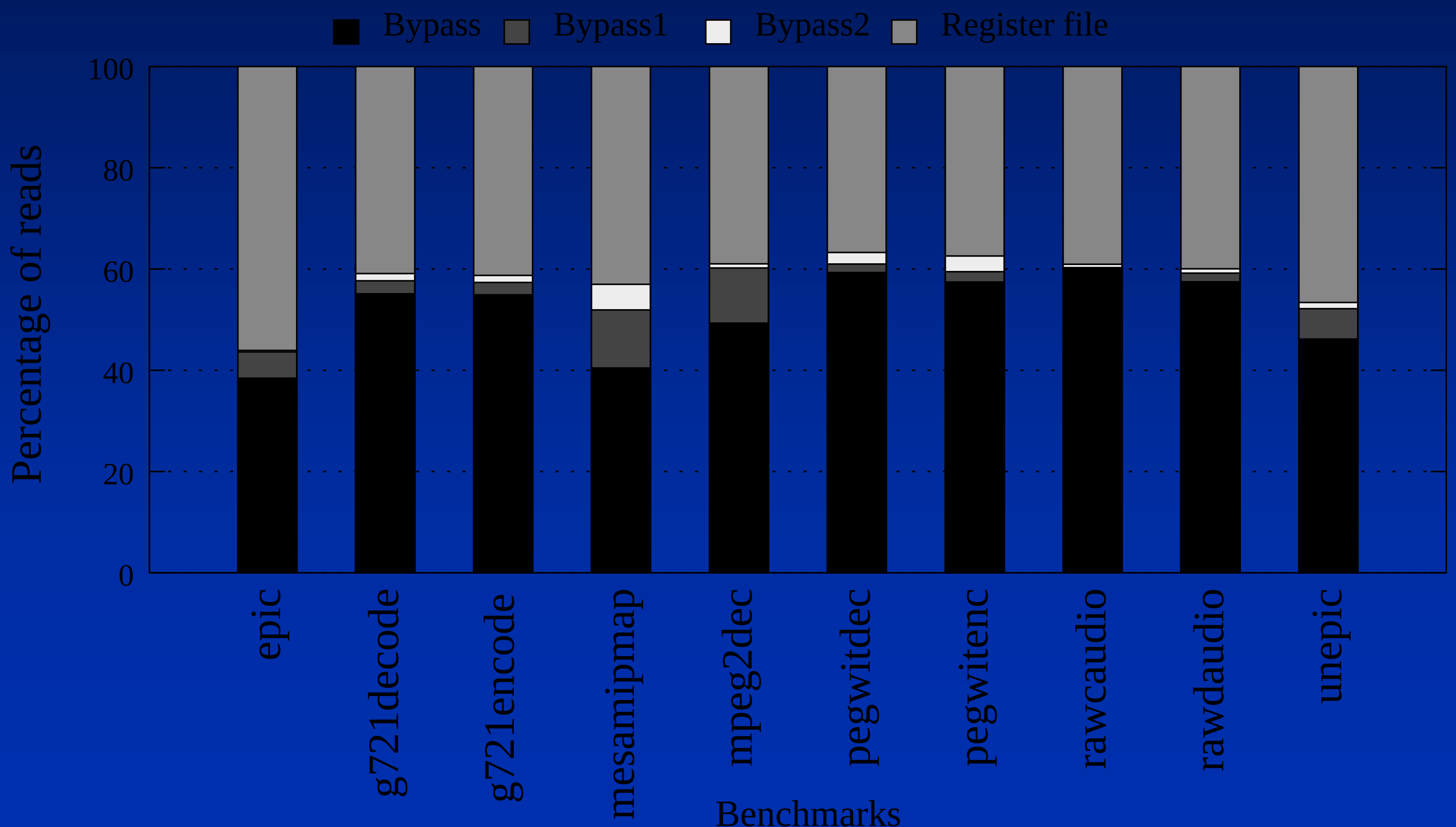
m is number of bypass sources

n is number of bypass destinations



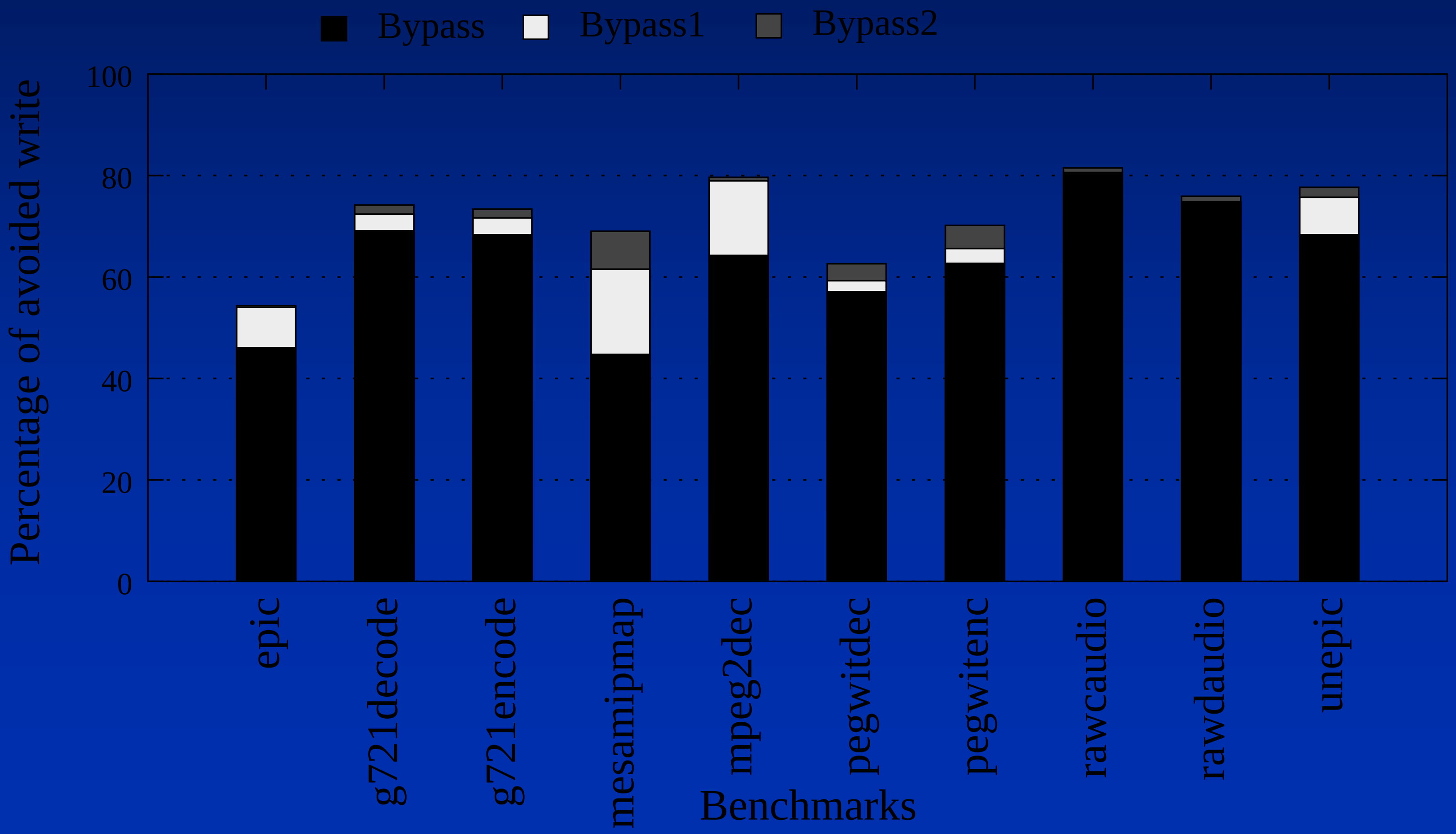
Experiments: Bypass reads

- Machine model
  - 2 Integer Unit, 2 Memory Unit, each single latency



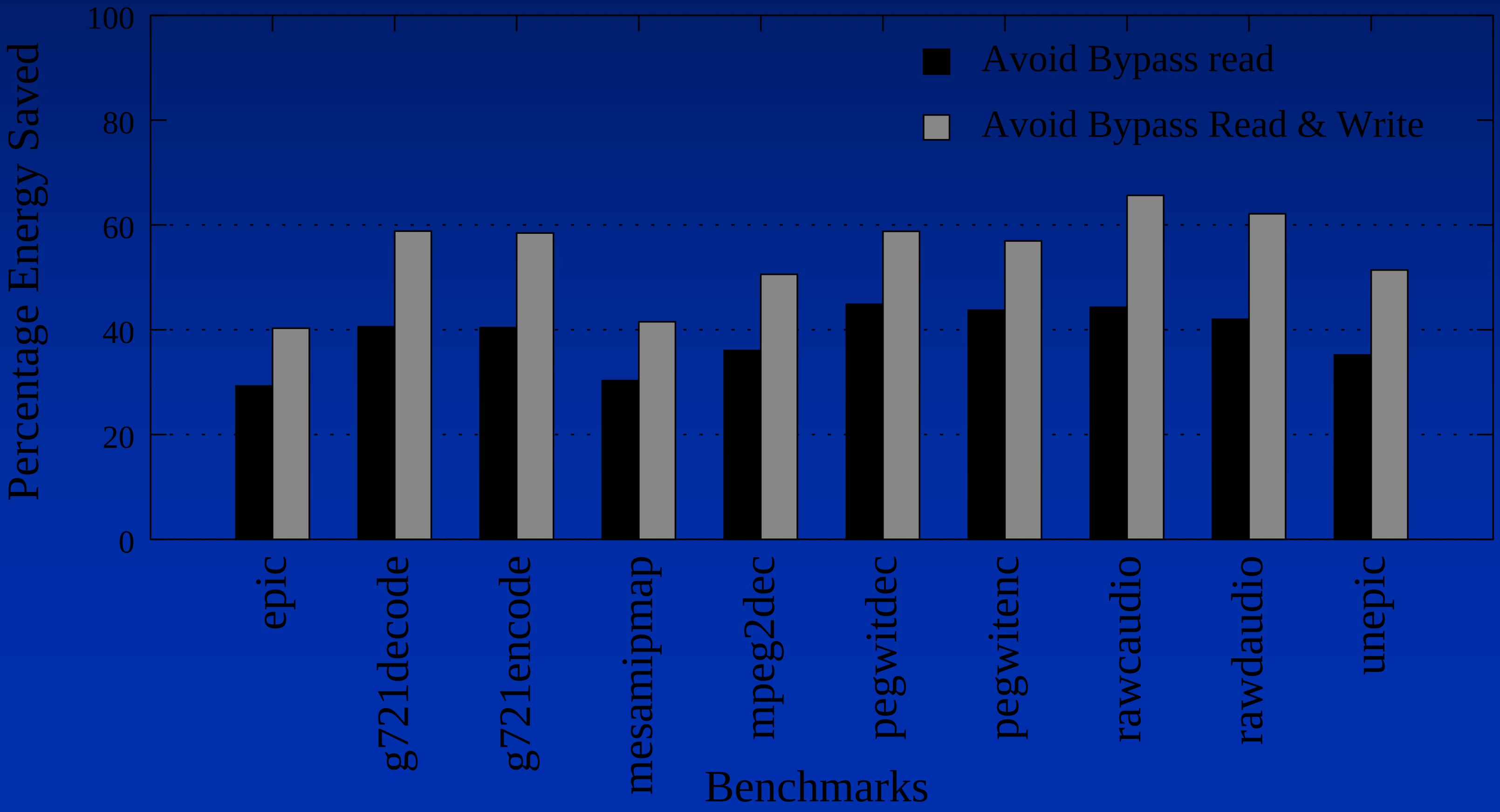
Experiments: Avoided writes

- Machine model
  - 2 Integer Unit, 2 Memory Unit, each single latency



Experiments: Energy Saved in RF

- Register File energy model
  - Cacti 4.0
  - 0.18μm, 64 bit, 64 registers



Issues and future work

- Approach is limited within basic block
  - No predication can be used
- Exception handling is more complicated
  - Need to store bypass registers as they are part of processor state
- A better register allocation algorithm
  - Which consider “last use” and “bypass” information to reduce register requirement

Conclusion

- Avoiding redundant reads and writes can save up to 40-60% energy of RF
- A novel approach of bypass control representation is suggested which leads to
  - Area savings of 3-5%
  - No increase in instruction width
- Multi-stage bypass can further increase the benefits

Questions
Thank you