



Fine Grained Multithreading Using Control/Data Partitioning

Aqeel Mahesri
CRHC



Outline

Motivation

Control and Data Threads

Execution Model

Performance

Future work

Conclusion



Motivation

High theoretical parallelism

Achievable parallelism limited by control dependence

Proposed solution: reduce time needed to calculate control flow

Reduced program determines control flow

2nd CPU executes remaining program



Control and Data Threads

Control thread

Subset of instruction stream including control instructions and all dependencies

Data threads

All remaining instructions

(optional) Dependencies of remaining instructions



Control and Data Threads

Control thread spawns data thread
after control decision

Each spawn is like a continuation

Small number of instructions

1-10 typical, average around 3

Control and Data Threads

Source Code

```
for (i = 0; i < n; i++)  
  a[i]++
```

Assembly Code

```
r3 ← 0 /* i */  
r4 ← n /* n > 0 */  
r5 ← base of a  
  
loop: sub r2 ← r4, r3  
      add r3 ← r3, 1  
      ld r6 ← 0(r5)  
      add r6 ← r6, 1  
      st r6, 0(r5)  
      add r5 ← r5, 8  
      bgt r2, loop  
  
done: ...
```

Control Thread

```
r3 ← 0 /* i */  
r4 ← n /* n > 0 */  
r5 ← base of a  
  
loop: SPANNDATA loop  
      sub r2 ← r4, r3  
      add r3 ← r3, 1  
      bgt r2, loop  
  
done: ...
```

Data Thread

```
r3 ← 0 /* i */  
r4 ← n /* n > 0 */  
r5 ← base of a  
  
loop: ld r6 ← 0(r5)  
      add r6 ← r6, 1  
      st r6, 0(r5)  
      add r5 ← r5, 8  
      ENDBLOCK
```



Partial Control Thread

Execution time of control thread is critical path

Some values produced in control thread not needed for a long time

Move these instructions to data thread

Communicate value from data processor to control processor

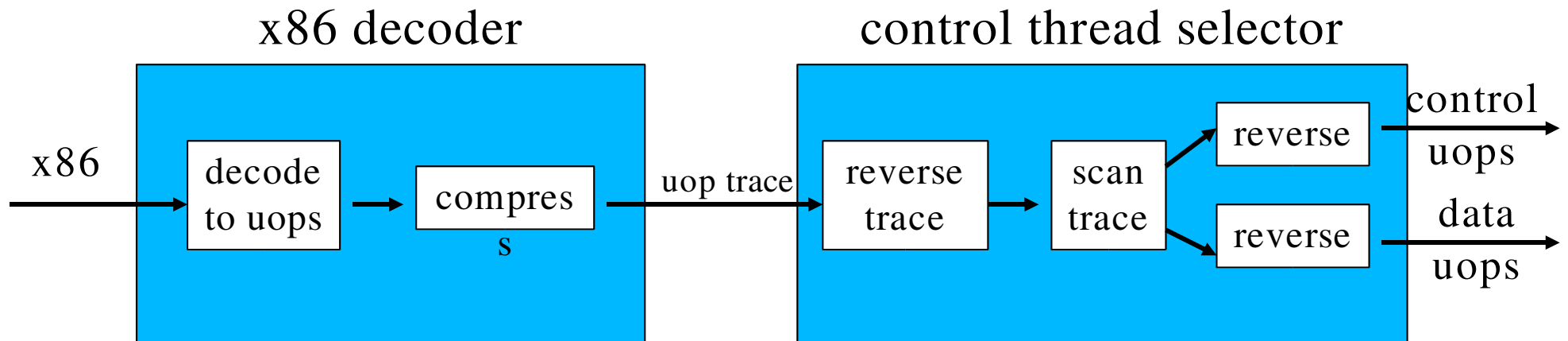
Trace Dependence Analysis

Control thread trace generator

Takes in uop trace decoded from x86 traces

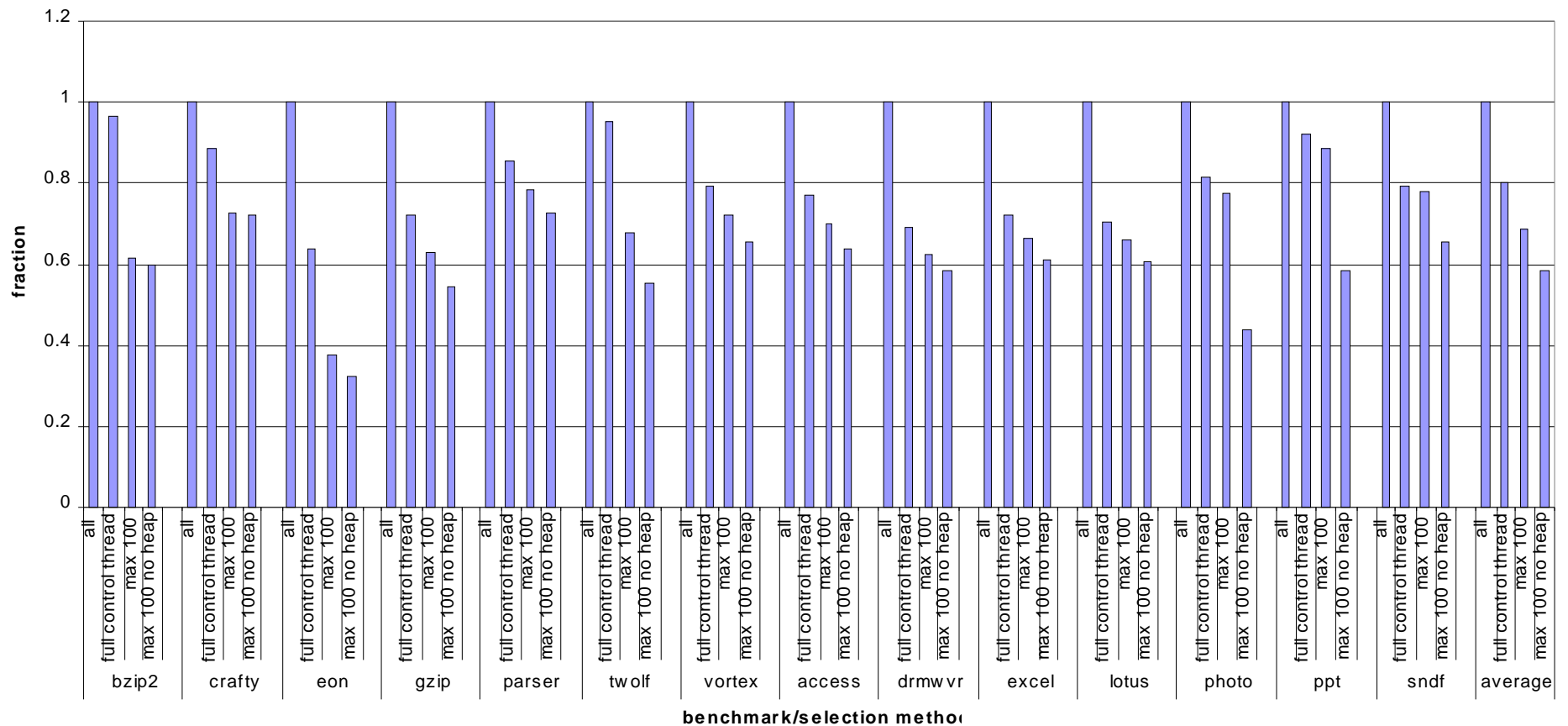
Classifies each uop as control or data

Writes control uop trace and data uop trace



Control Thread Size

Fraction uops in control thread





Execution Model

Dual-core CMP

One core executes control thread,
the other the data thread

Communication of control flow
information from control to data
processor

Communication of register and
memory values



Simulation Infrastructure

Hacked sim-beta

- Simulates dual-core CMP

- Simulator forks second sim-beta process

 - Original process simulates control, new process data

 - Interprocessor communication with Unix sockets

 - Control sim-beta manages shared state

- 5 cycle communication latency

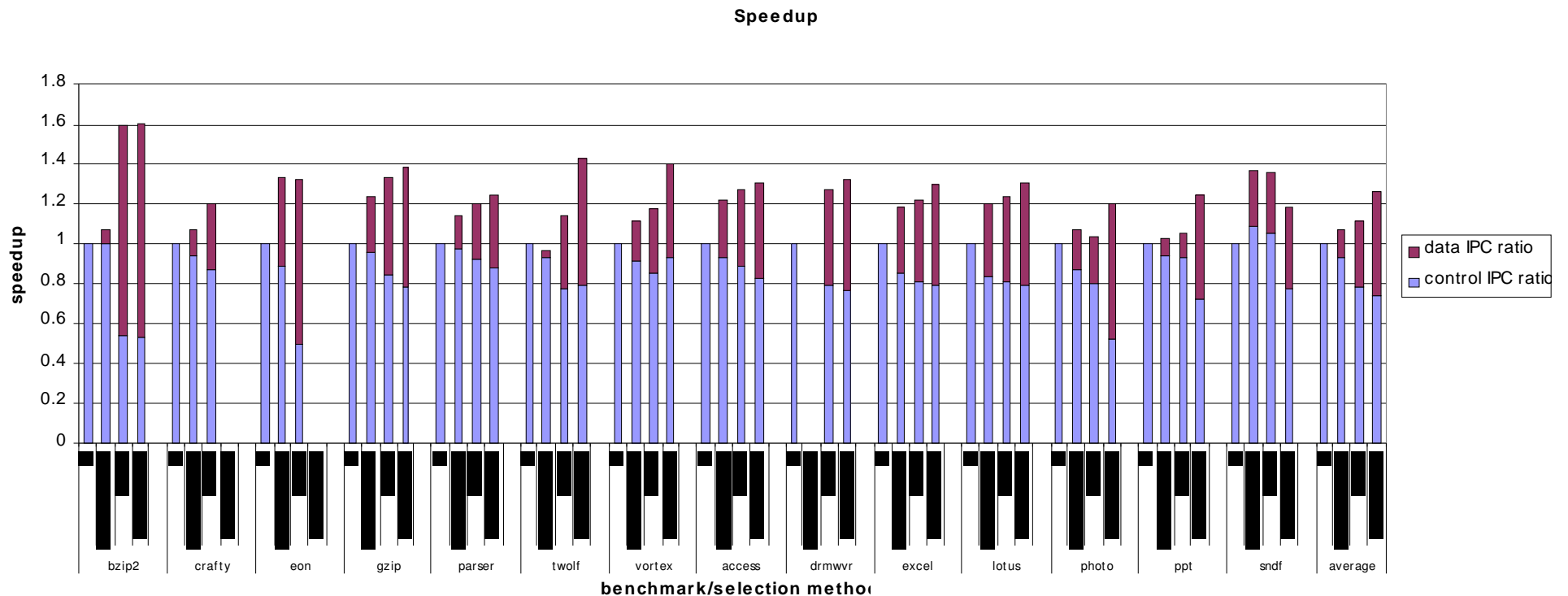
- Models delays from misspeculation in control

 - But does not model incorrect execution path

Performance Results

Speedup of 7% when running full control thread

Speedup rises to 27% with limited dependency distance and some memory dependencies removed





Analysis

Control stream is critical path

Data thread stalled most of the time

Performance falls short of uop ratio
by 30%

Mostly due to delays from
misspeculation

Possibly smaller in real system due to
prefetching effects

Density of data dependences



Analysis (cont.)

Simulation glosses over many issues

Control thread selection

how close is trace dependency analysis to what a real system would do?

Synchronization

currently handled by embedding meta data in traces



Future Work

Further reduce control thread

move dependencies of highly biased branches to data processor

what else?

Wider parallelism

Use advance knowledge of control flow to spread data computation over more processors



Conclusion

Control flow major bottleneck limiting performance

Extract smaller control thread from full program

Use to compute control flow faster

Data computation on second, lagging processor

Significant performance benefit

7% to 27% average speedup depending on partitioning method

Significant challenges