CSC2/458 Parallel and Distributed Systems
Automated Parallelization in Software

Sreepathi Pai

January 30, 2018

URCS

Outline

Out-of-order Superscalars and their Limitations

Static Instruction Scheduling

Outline

Out-of-order Superscalars and their Limitations

Static Instruction Scheduling

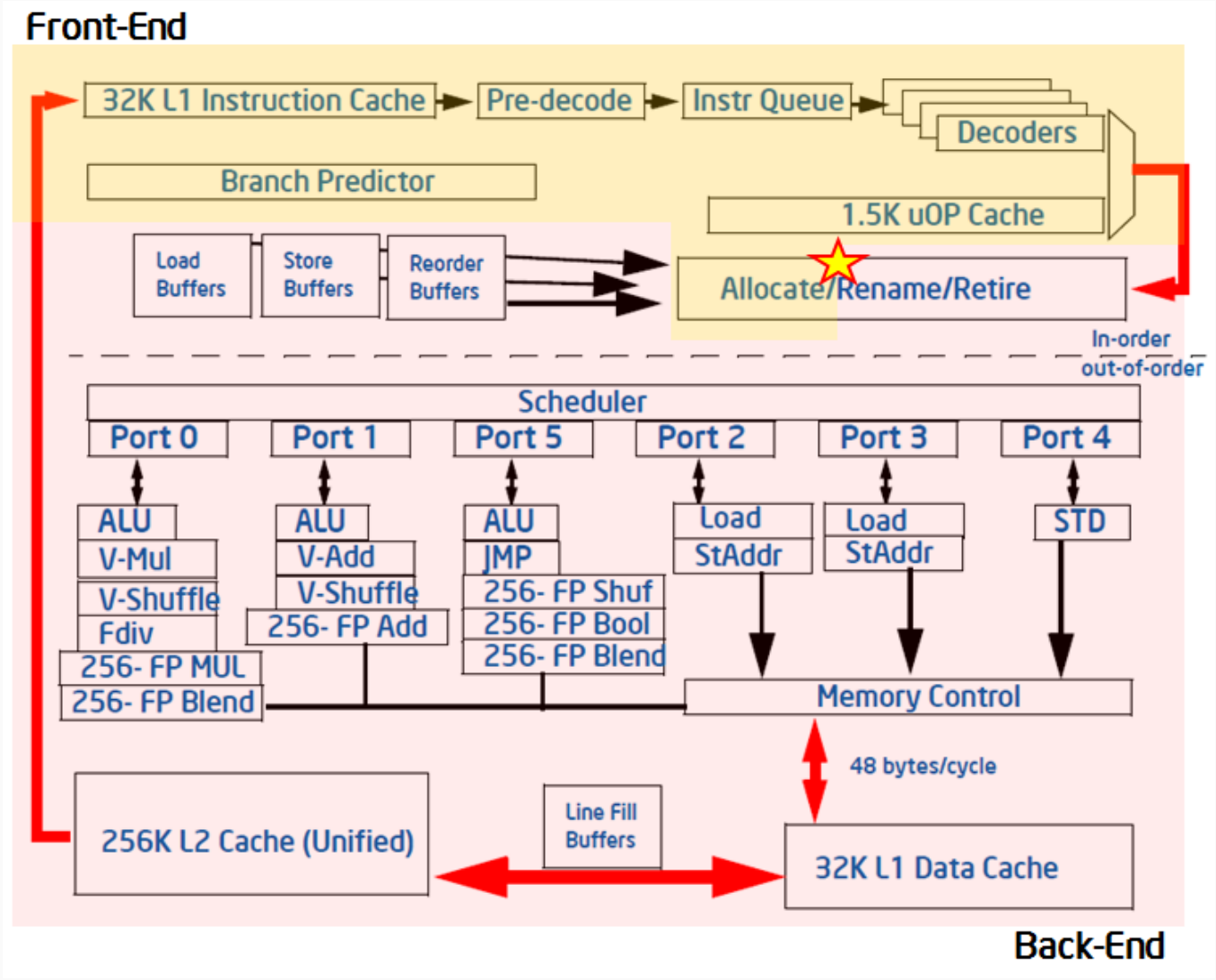
How will a processor parallelize this?

```
for(i = 0; i < A; i++) {
   sum1 = sum1 + i;
}

for(j = 0; j < A; j++) {
   sum2 = sum2 + j;
}
```

Dynamic Instruction Stream

```
i = 0
i < A (true)
sum1 = sum1 + 0
i++
i < A (true)
sum1 = sum1 + 1
i++
 ...
i < A (false)

j = 0
j < A (true)
sum2 = sum2 + 0
j++
j < A (true)
sum2 = sum2 + 1
j++
j < A (true)
 ...
```

An Intel Processor Pipeline



Source: Intel

Instruction Pipeline

- Instructions flow into "issue window"
  - from dynamic instruction stream
- Dependences are calculated and resources allocated
- Independent instructions are dispatched to backend *out-of-order*
- Instructions are *retired in-order* using a "reorder buffer"

Outline

VLIW Processors

- Very Long Instruction Word Processors
- Can execute multiple instructions at the same time
  - So superscalar
- But leaves independence checking to the compiler
  - Compiler packs instructions into "long words"
- Example:

	Slot 1	Slot 2
VLIW1:	ins1	ins2
VLIW2:	ins3	[empty]

VLIW example

Consider *static code* below:

```
for(i = 0; i < A; i++) {
   sum1 = sum1 + i;
}

for(j = 0; j < A; j++) {
   sum2 = sum2 + j;
}
```

For a 2-wide VLIW, one packing could be:

Slot 1	Slot 2
`i = 0`	`j = 0`
`i < A`	`j < A`
`sum1 = sum1 + i`	`sum2 = sum2 + j`
`i++`	`j++`

Program Semantics

- When processors commit in-order, they preserve appearance of executing in program order
  - Not always true when multiple processors are involved
- But when compilers emit code, they change order from what is in program
- Which orders in the original program must be preserved?
- Which orders do not need to be preserved?

Our Ordering Principles

- Preserve Data Dependences
- Preserve Control Dependences

What about:

```
printf("hello");
printf("world");
```

Basic Block Scheduling

- Basic block is a single-entry, single-exit code block
- Instructions in basic block have the same control dependence
  - All can execute together if they have no dependence
- Is there an advantage in reordering instructions within a basic block?

Instruction Scheduling

Consider:

```
A = 1       // takes 1 cycle
B = A + 1   // takes 1 cycle
C = A * 3   // takes 2 cycles and 2 ALUs
D = A + 5   // takes 1 cycle
```

Assume you have 2 ALUs. How should you schedule these instructions to lower total time?

Increasing the size of Basic Blocks

- Basic blocks are usually small
  - Not many opportunities to schedule instructions
- How can we increase size of basic blocks?
  - Remember out-of-order processors do speculation ...