IA-64 / VLIW from a Compiler Optimization Perspective

Frédéric de Mesmay

Theodoros Strigkos

03/01/2007

Papers

- “OS and Compiler Considerations in the Design of the IA-64 Architecture”

  Rumi Zahir, Jonathan Ross, Dale Morris and Drew Hess

  ASPLOS 2000

- “A Comparison of Full and Partial Predicated Execution Support for ILP Processors”

  Scott A. Mahlke, Richard E. Hank, James E. McCormick, David I. August and Wen-Mei W. Hwu

  ISCA 22

What makes VLIW tick?

- Speculation
  - Control Speculation
  - Data Speculation
- Predication

What can the compiler do to make the hardware simpler and faster?

Control Speculation

- Issue long-latency instructions before we now for sure that we need them
  - e.g. loads
- Example

```
if (x == 0) {
  load r1, a;
  use r1;
}
else r1 = 0;
```

Speculative Load

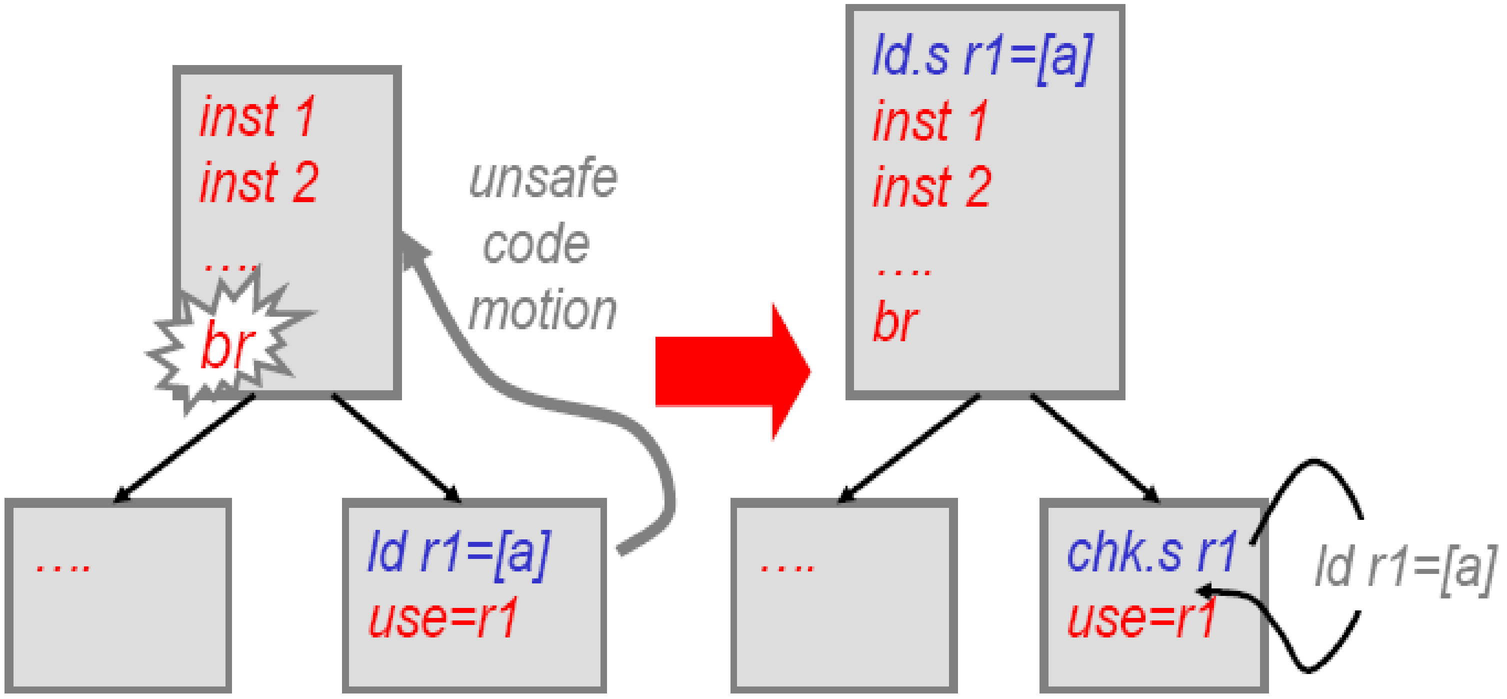
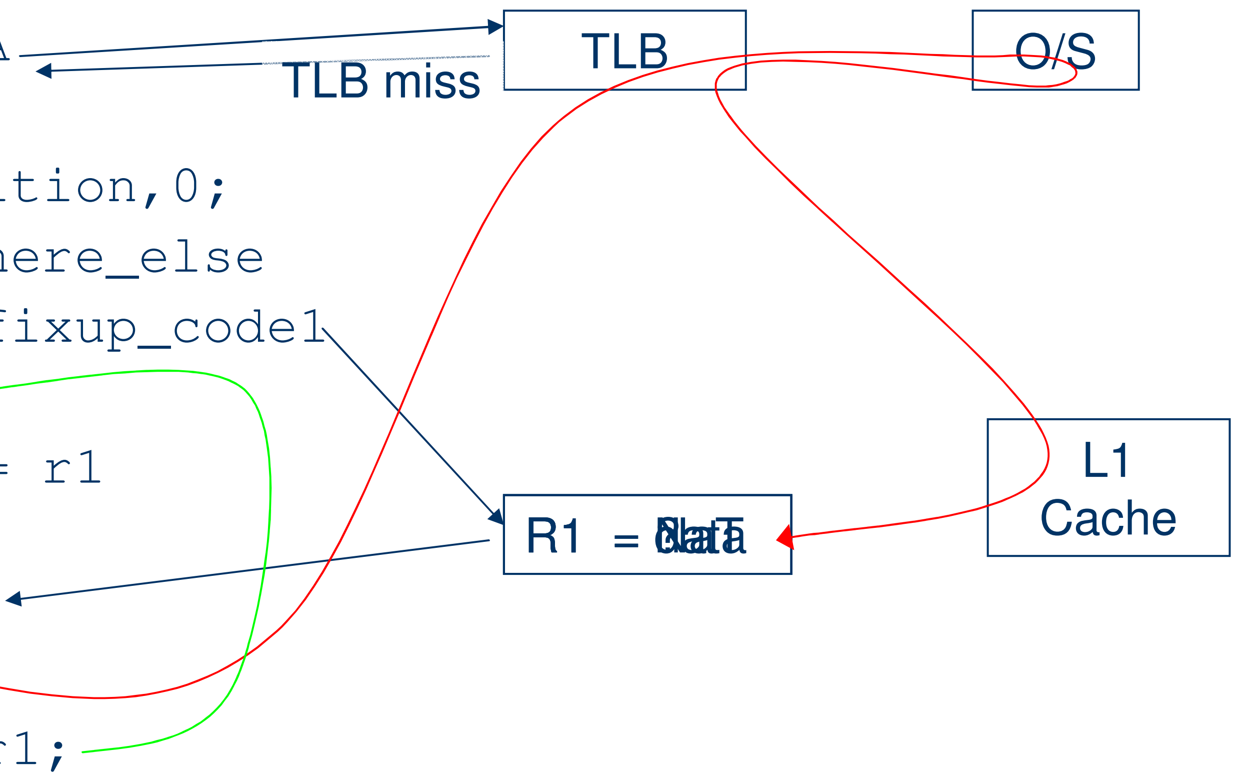


Figure from B. Falsafi

Control Speculation
Eager Deferral – With Recovery Code

- Recovery Model

```
   ld4.s r1,A
...
   cmp Rcondition,0;
   bne somewhere_else
   chk.s r1,fixup_code1
reenter1:
   st4 [r2] = r1
...
fixup_code1:
   ld4 r1,A
   br reenter1;
```

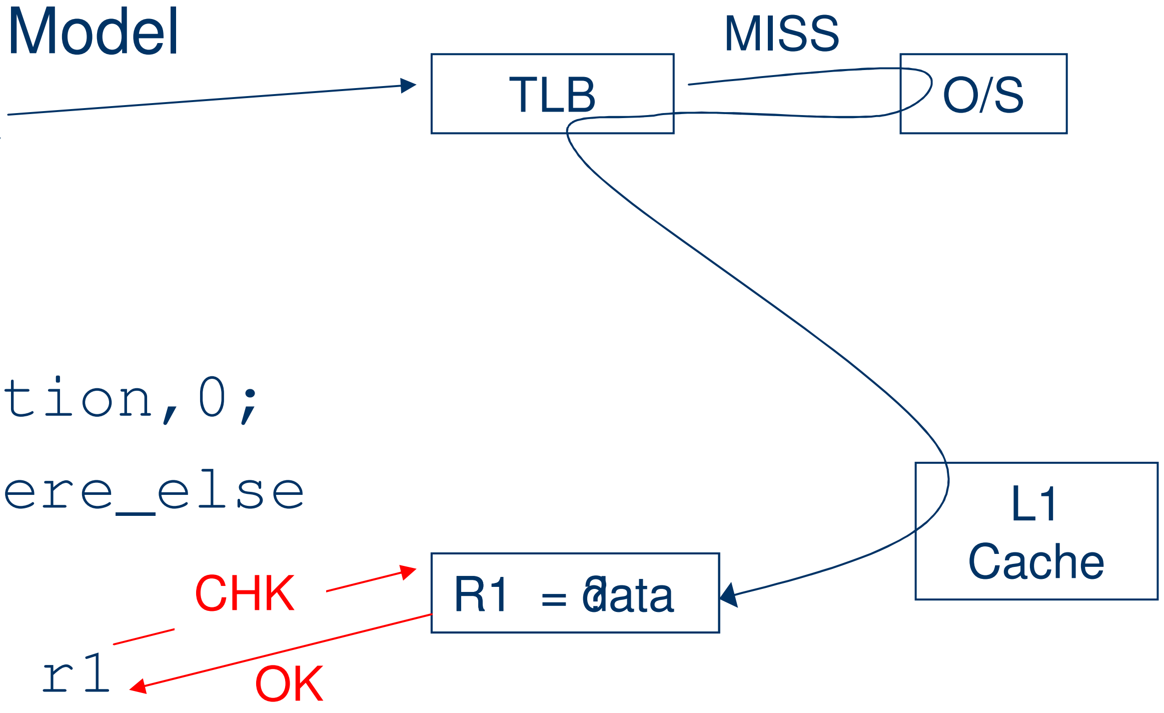


Control Speculation
Minimal Deferral – No Recovery Code

- No Recovery Model

```
  ld4.s r1,A
  add   r1,7
...
  cmp Rcondition,0;
  bne somewhere_else

  st4 [r2] = r1
...
```



Data Speculation

- Issue long-latency loads fast without knowing if a conflicting store follows (aliasing problem)
- Example:

```
foo (char *a, int *p) {
   *a = 1;
   b = *p + 5;              // p probably != a
   ...
}
```

  - Load p before calculating the target of a.
  - Upon mis-speculation, re-load p

Data Speculation

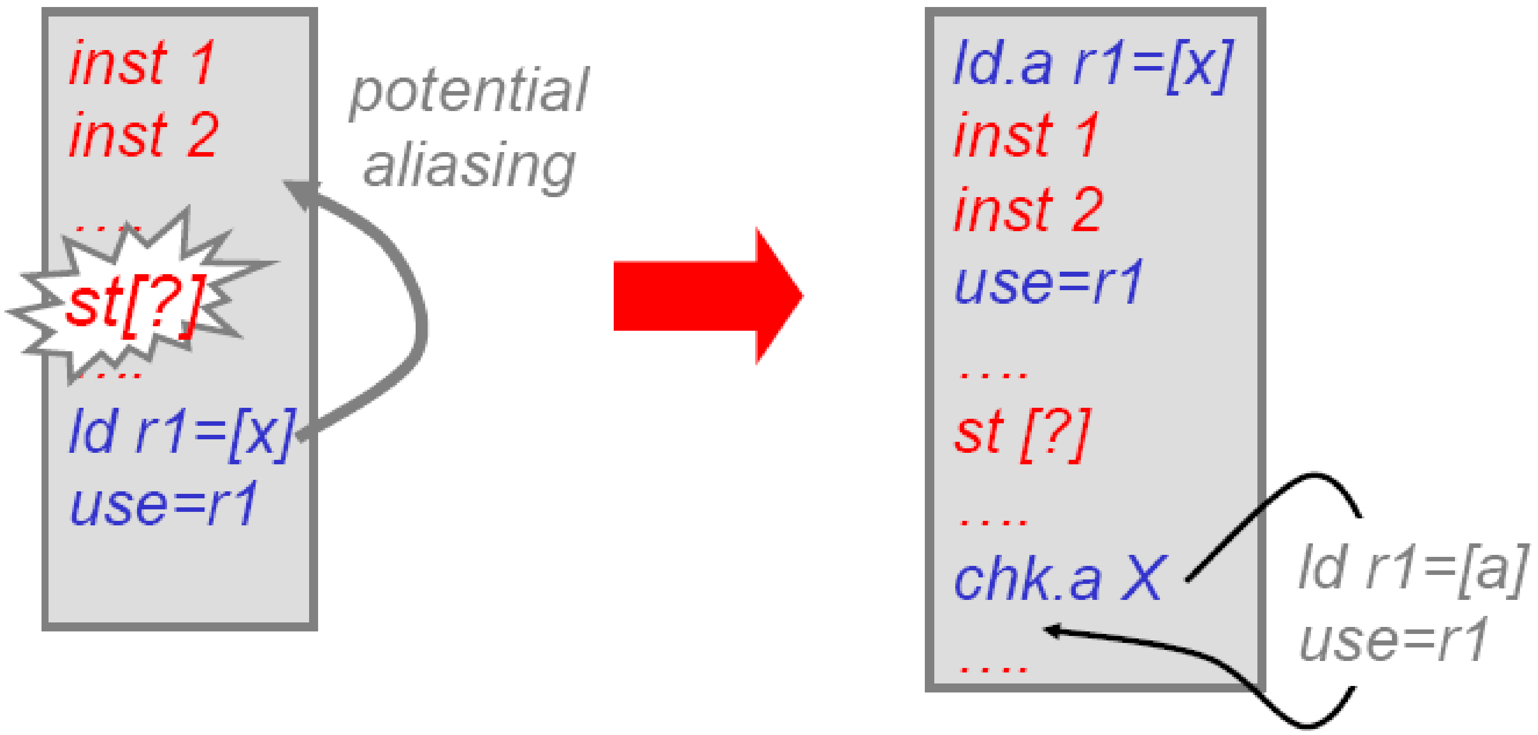
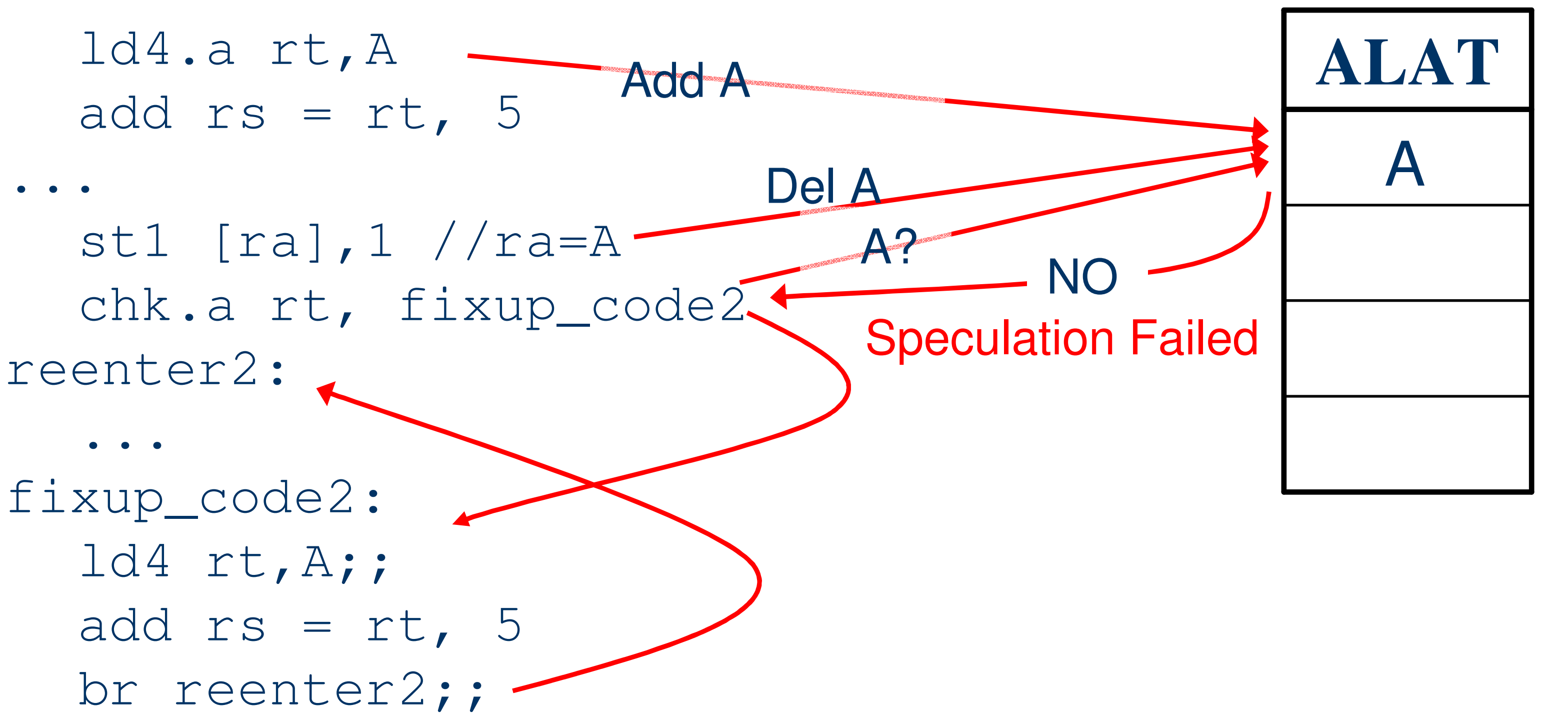


Figure from B. Falsafi

Data Speculation
IA-64 Advance Load Address Table

- When an advanced load is issued, add target in the ALAT
- All stores delete conflicting ALAT records
- The chk.a instruction searches for the ALAT record
  - If found, speculation was correct; otherwise, reload
- Easy to implement and handle context-switch
  - If ALAT full, or context-switch, drop entries and cause recovery

Advanced Load Example



```
   ld4.a rt,A
   add rs = rt, 5
...
   st1 [ra],1 //ra=A
   chk.a rt, fixup_code2
reenter2:
   ...
fixup_code2:
   ld4 rt,A;;
   add rs = rt, 5
   br reenter2;;
```

Predication

- Execute both paths and decide which is correct
  - Reduces control hazards
- Comes in two flavors:
  - Chocolate: Full predication
  - Vanilla: Partial predication

Predication

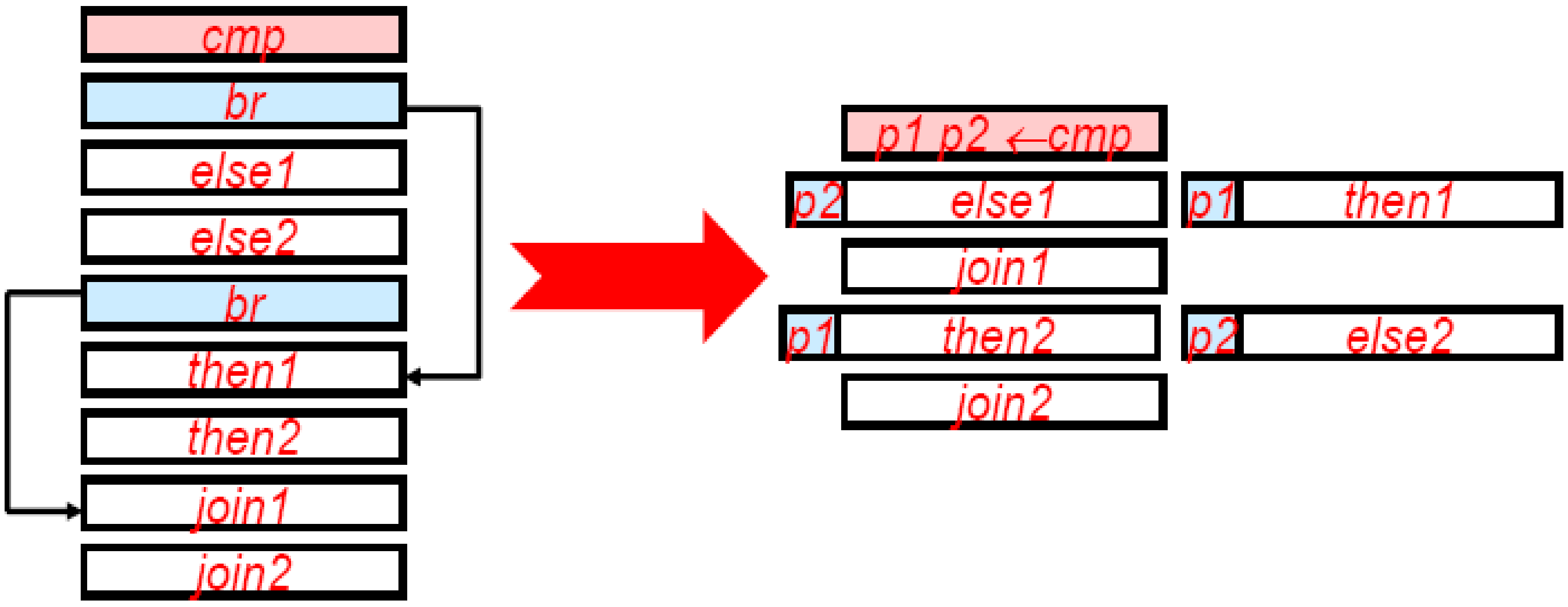


Figure from B. Falsafi

Full Predication

- All instructions may have a predication input
  - `add rdest,r1,r2 (Pin)`

- Predicate Define instructions

  `pred` $_{\texttt{<cmp>}}$ `Pout1`$_{\texttt{<type>}}$`, Pout2`$_{\texttt{<type>}}$`, src1, src2 (Pin)`

  - `Type = U, U_bar, OR, NOR, AND, NAND`

Partial Predication

- Only conditional moves or selects
  - `CMOV dest,src,cond` $\rightarrow$ `if (cond) dest = src;`
  - `Select dest,src1,src2,cond` $\rightarrow$
    `dest = cond?src1:src2`
- Usually combined with predicate promotion
  - Execute most instructions with no predicate and conditionally move final results

Chocolate or Vanilla?

Full Predication

+ Fewer instructions
+ Faster Program
+ Decreases register pressure

- Significant ISA changes
- More register file ports required

Partial Predication

+ Easier to adapt ISA
+ Few instructions have 3rd source operand

- Predicate Promotion

→ useless instructions are executed

→ register pressure increased

Is it worth it?

- VLIW appeared as the opposite pole of OoO cores: Complicated S/W to relax H/W.
  - Predicates in all instructions instead of prediction
  - ALAT instead of load-store queue
  - Speculative instructions instead of OoO issue
  - Register Stack Engine as opposed to renaming
  - …

Questions?

- Thank you!