Optimizing IA-64 Software Performance

Jim Pierce
Gary Carleton
Microprocessor Software Labs
Intel Corp.

Intel Developer Forum Spring 2000

Intel Labs

Keys to High Performance

- Great IA-64 features - how can I get them?
- Compiler will do it automatically
- Spend your time helping the compiler to do even better

Agenda

- Good right out of the box
  - Three examples
- How you can make it even better
  - Profile Feedback
  - Alias Analysis
  - Interprocedural Analysis

IA-64 High Performance Compiler

- Full support for architecture features
  - predication, speculation, and software pipelining
- Multi-file capability
  - inlining, alias analysis, interprocedural optimizations
- Profile-guided optimizations (PGO)
- Memory hierarchy optimizations
  - loop transformations, prefetch, scalar replacement (-Qhlo switch)

Compiler On Its Own #1

```
int livecar(NODE *n)  {
   switch (((n)->n_type)) {
        case 7:   case 10:
          vmark(n);
        case 1:   case 2:  case 5:
        case 9:   case 6:   case 8:
          return (0);
         case 4:  case 3:
         …
    }
}
```

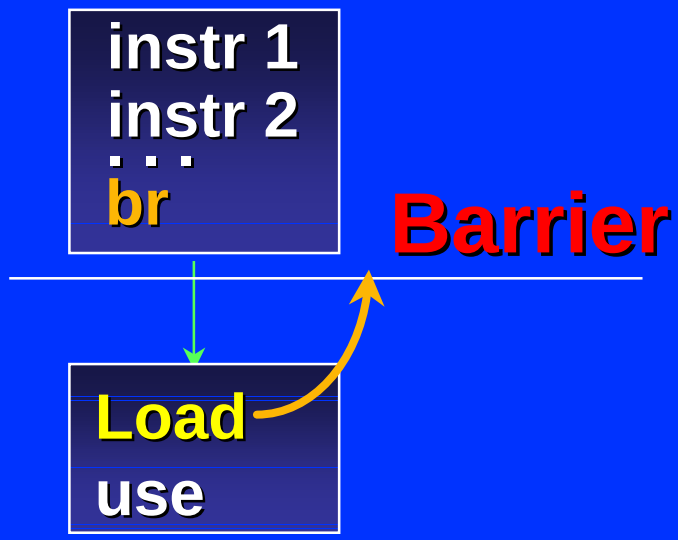
Hot function in Spec95 li

Common example of switch statement

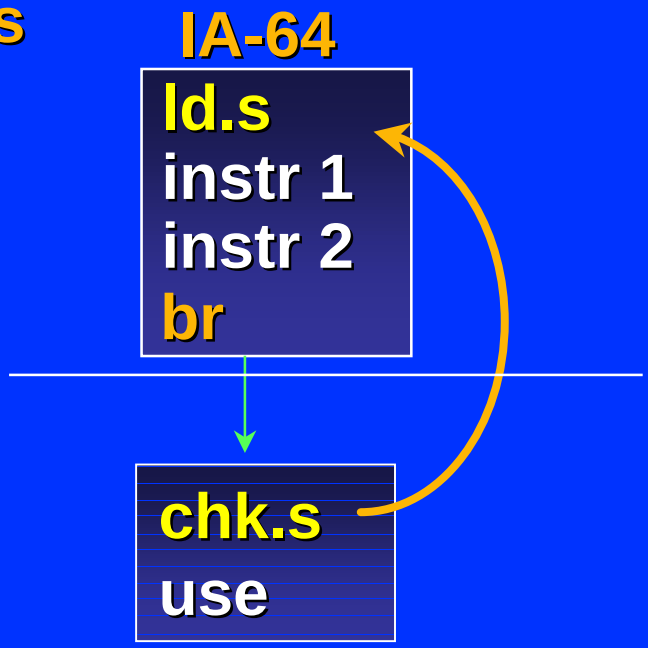
Good use of predication and speculation

Speculation Review

Traditional Architectures



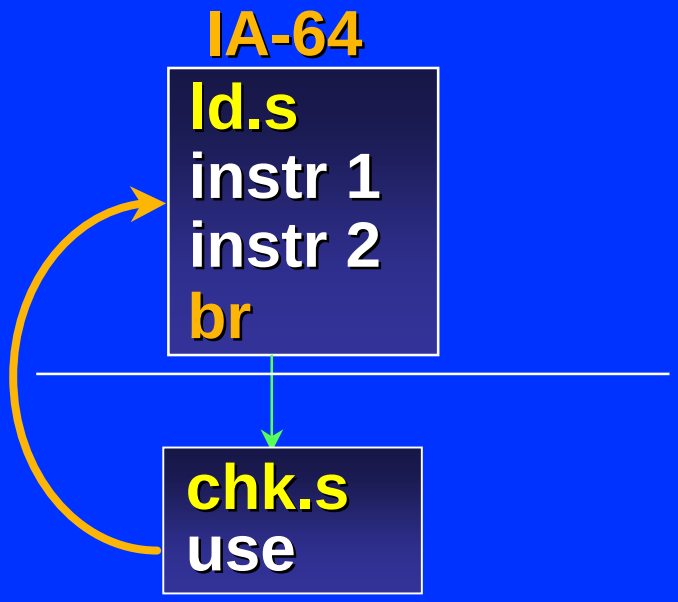
IA-64



Allows elevation of load, even above a branch

Speculation hides memory latency and increases available parallelism

Hoisting Uses



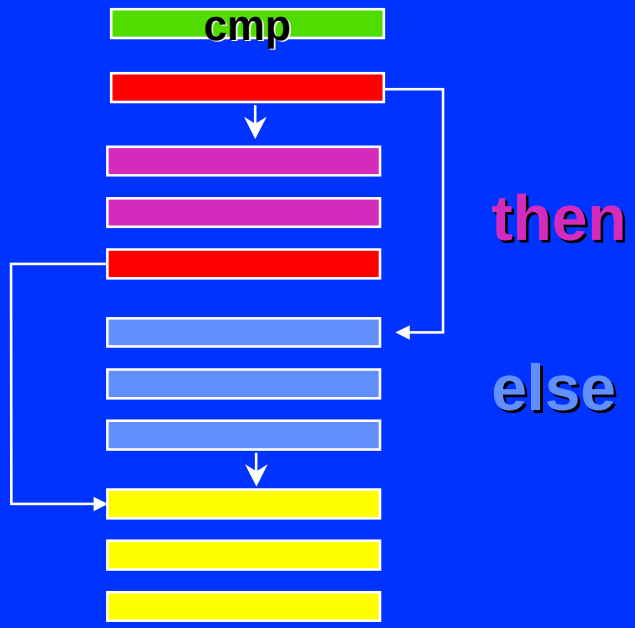
- The uses of speculative data can also be executed speculatively
  - distinguishes speculation from simple prefetch

Enables Further Parallelism

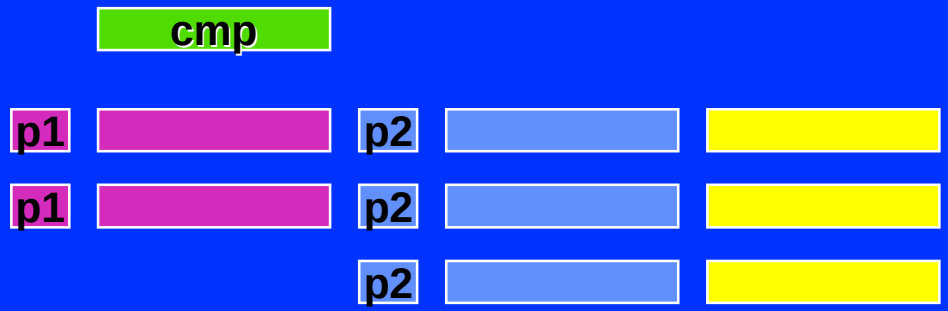
Predication Review

Example #1

Traditional Architectures



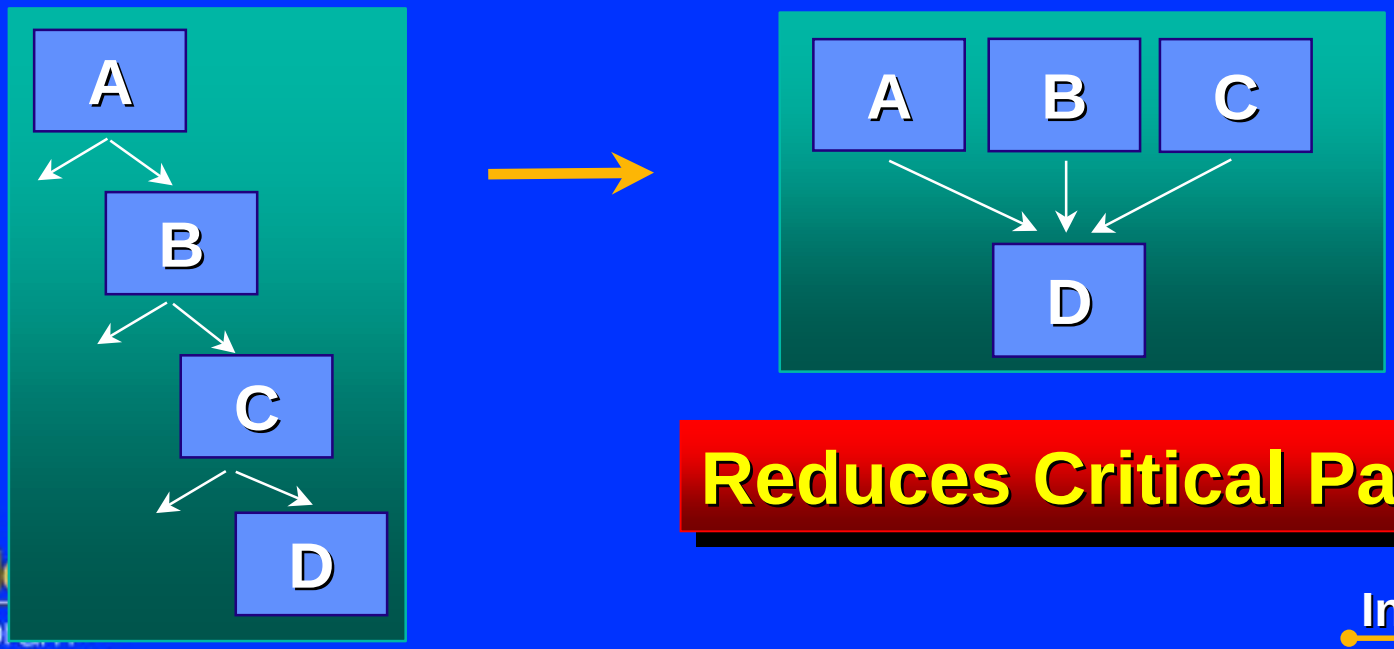
IA-64



- Removes branches, converts control flow into data flow
  - Executes multiple paths simultaneously
- Increases performance by exposing parallelism and reducing critical path
  - Removes poorly predicted branches
  - Better utilization of wider machines

Parallel Compares

- Three types:
  - AND: both target predicates set FALSE if compare is false
  - OR: both target predicates set TRUE if compare is true
  - ANDORCM: if true, sets one TRUE, sets other FALSE



Reduces Critical Path

Parallel Compares

Or Predicate

- Predicate is initially cleared
- Any true compare will set
- Any false compare does nothing

```
0    cmp.unc.ne  p1 = r0,0
1    cmp.or      p1 = cond1
     cmp.or      p1 = cond2
```

And Predicate

- Predicate is initially set
- Any true compare does nothing
- Any false compare will clear

```
0    cmp.unc.eq  p1 = r0,0
1    cmp.and     p1 = cond1
     cmp.and     p1 = cond2
```

Reduces critical path

Recall code example

```
int livecar(NODE *n)  {
   switch (((n)->n_type)) {
       case 7:   case 10:
          vmark(n);
       case 1:   case 2:  case 5:
       case 9:   case 6:   case 8:
          return (0);
        case 4:  case 3:
        …
    }
}
```

Compile without profile information

Livecar Assembly Code

```
livecar:                                                              (cycle)
      alloc   r33=ar.pfs,1,6,2,0     ld1     r37=[r32]                    1
      mov   r39 = r32                mov   r8 = r0                        1
      cmp.ne    p1,p2=r0,r0          cmp.ne  p3,p0=r0,r0                  2
      cmp.eq.orand  p1,p2=7,r37  cmp.eq.orand    p1,p2=10,r37             3
(p1)  br.call          vmark                                             3
(p2)  cmp.eq.or  p3,p0=5,r37     (p2)  cmp.eq.or  p3,p0=1,r37            4
(p2)  cmp.eq.or  p3,p0=8,r37     (p2)  cmp.eq.or  p3,p0=9,r37            4
(p2)  cmp.eq.or  p3,p0=6,r37     (p2)  cmp.eq.or  p3,p0=2,r37            5
(p3) br.ret                                                             5
(p2) ....                                                               6
```

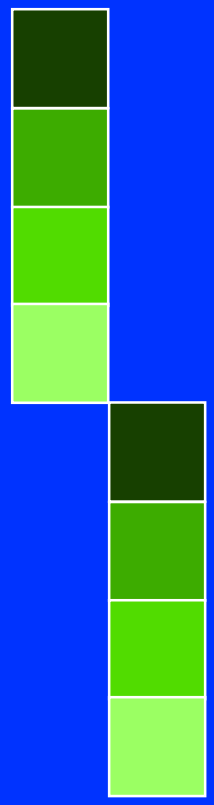
Compiler On Its Own #2

```
void foo(int *x, int *y, int *z) {
    int i;
    for (i = 0; i < 100; i++)
        z[i] = A * x[i] + y[i];
}
```

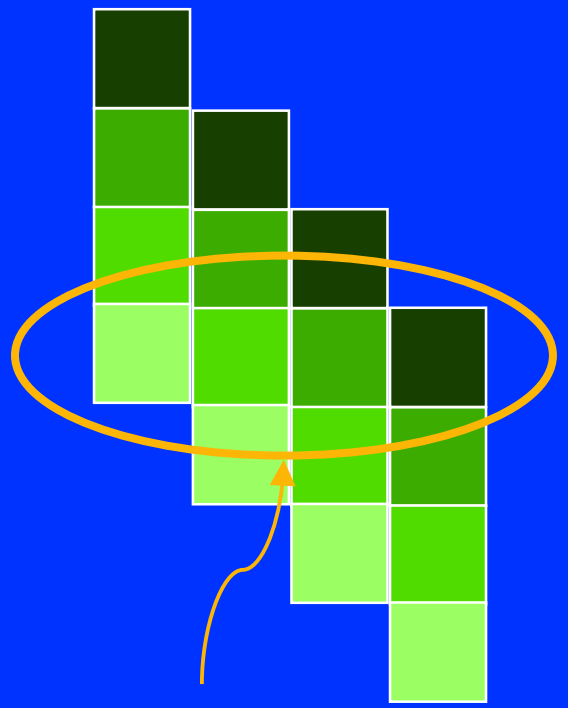
- Example of use of
  - Software Pipelining
  - Data Speculation
- x, y and z cannot be disambiguated

Software Pipelining

Allows overlapping execution of multiple loop iterations - more iteration in same amount of time



vs.



Whole loop computation in one cycle

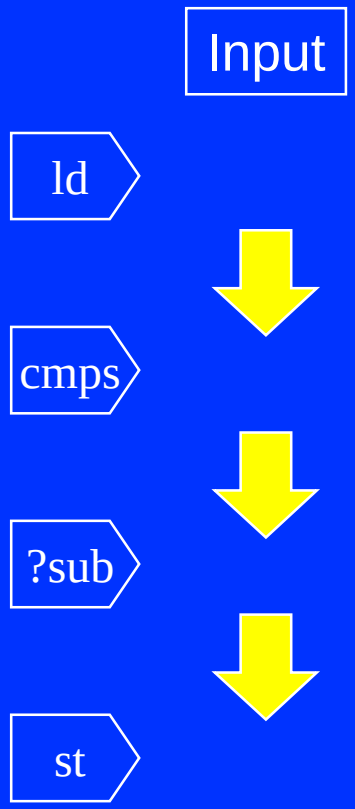
Software Pipelining

Example #2

Case conversion loop

```
for (i=0, i< len, i++) {
    if (line[i] >= ‘a’ && line[i] <= ‘z’)
        newline[i] = line[i]-32;
    else
        newline[i] = line[i];
}
```

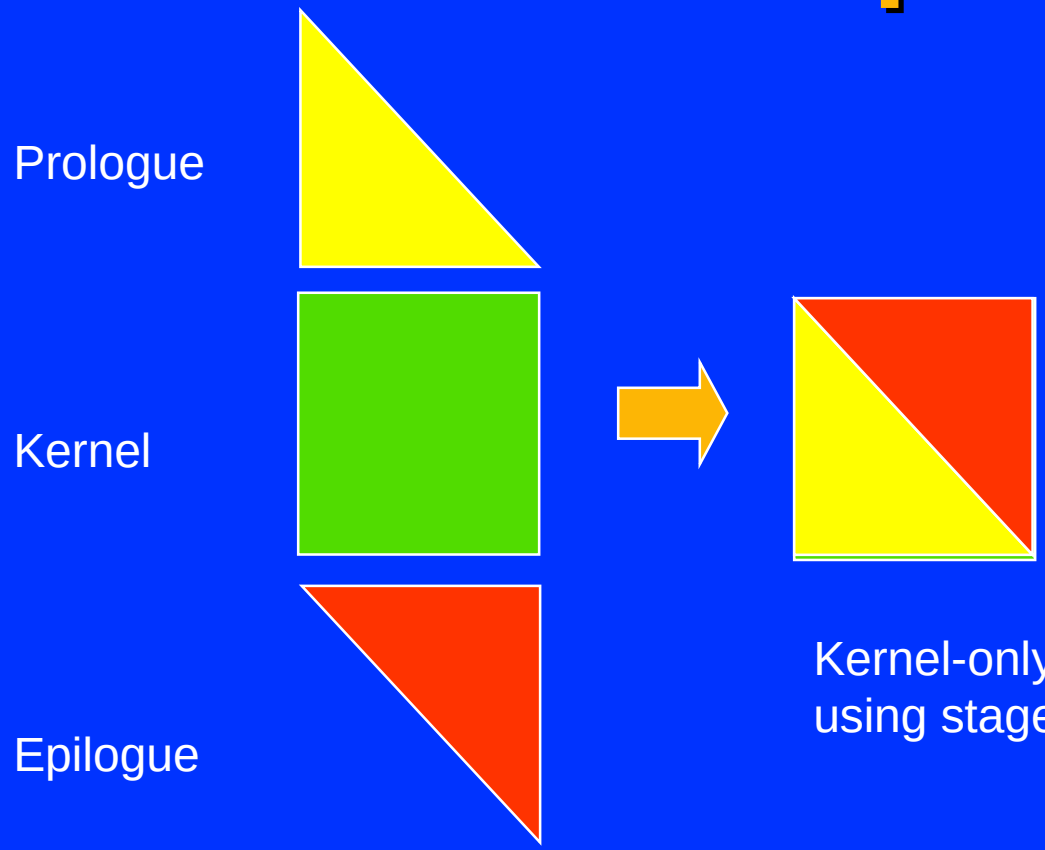
Cycle



Intel Developer Forum Spring 2000

Intel Labs

Software Pipelining



Kernel-only code
using stage predicates

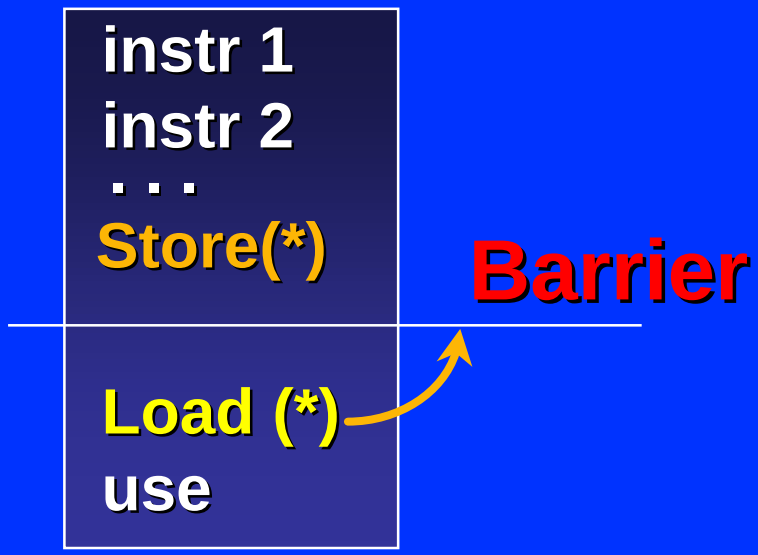
IA-64 Features

- Rotating Registers
- Loop branches
- Full predication
- Rotating Predicates

SWP applicable to many loops in IA-64

Data Speculation Review

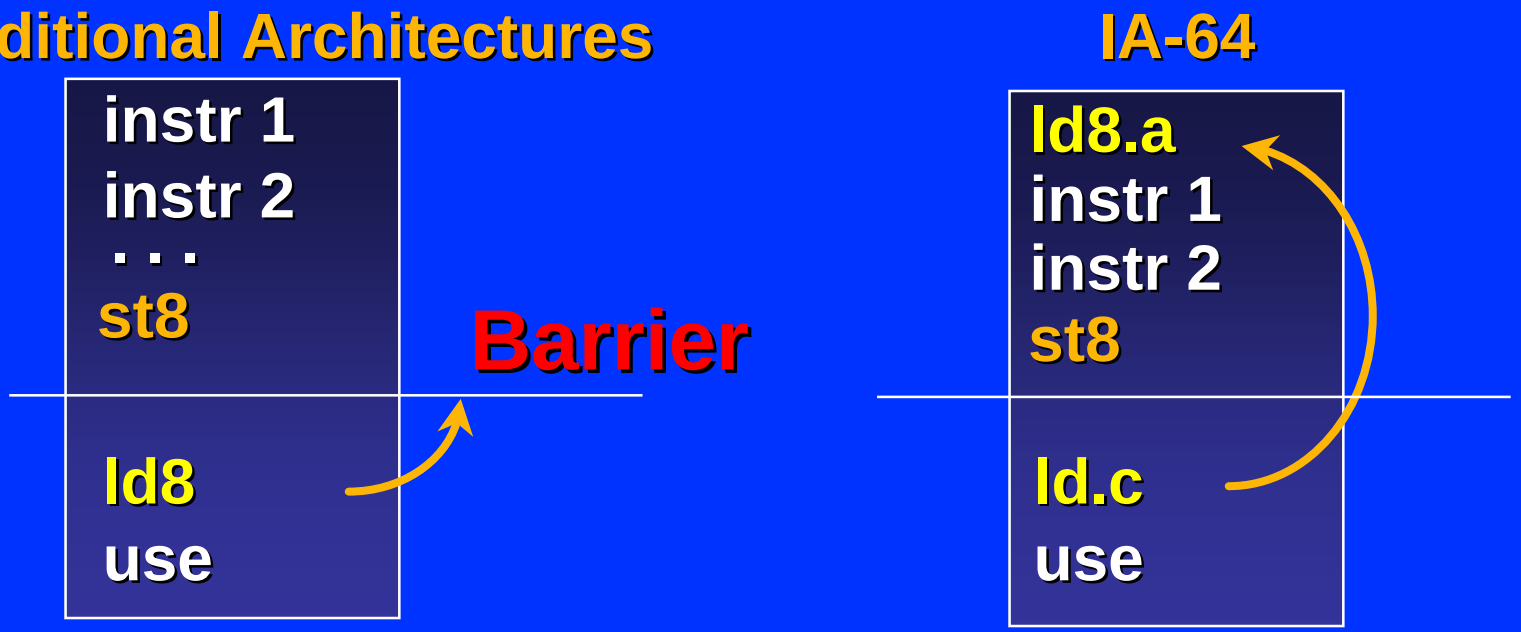
Traditional Architectures



Often compiler cannot hoist load over store because of limited alias knowledge

Data Speculation Review

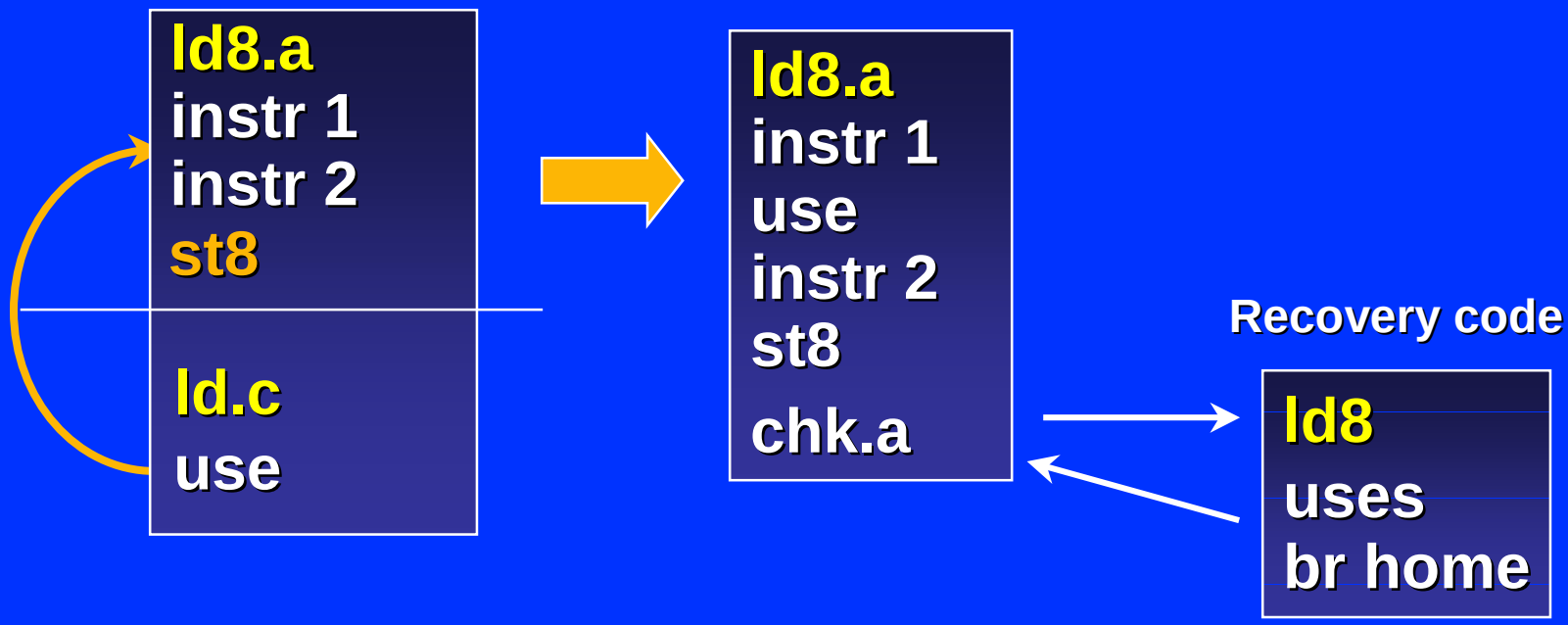
In IA-64, compiler can issue a load prior to a preceding, possibly-conflicting store



Unique feature to IA-64

Data Speculation

Uses can also be hoisted



Synergy with control speculation yields greater performance

Back to Example #2

```
void foo(int *x, int *y, int *z) {
    int i;
    for (i = 0; i < 100; i++)
        z[i] = A * x[i] + y[i];
}
```

- Compiler automatically uses data speculation with software pipelining
- No SWP - 24 cycles/iteration
- SWP and data spec. - 4 cycles/iteration

Real Application

- Similar loops found in ijpeg video compression benchmark
- SWP and data speculation together
  - 33 -> 21 cycles in h2v2_smooth_downsample
  - 35 -> 8 cycles in fullsize_smooth_downsample

Compiler On Its Own #3

Code segment from a real world multimedia application
Compiler added prefetch instructions

```
for i = 1, M
  read *(src++);
  work
  write *(dst++)
end_for
```

```
for i = 1, M
  if (i mod 4 == 0) {
    lfetch(src+d)
    lfetch(dst+d)
  }
  read *(src++)
  work
  write *(dst++)
end_for
```

Using -Qhlo option

Speedup: Intrinsics 60%
Compiler 140%

Intel Developer Forum Spring 2000

Intel Labs

Compiler Prefetching

- Advantages
  - Automatic and error-free
    - Computes distances based upon datasizes and microarchitecture
  - Aware of other optimizations
    - software pipelining
  - Exploit architecture features
    - predication, rotating registers
  - Source remains portable

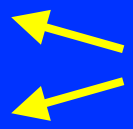
Agenda

- Good right out of the box
  - Three examples
- How you can make it even better
  - Profile Feedback (PGO)
    - What to do if you don't have profile?
  - Alias Analysis
  - Interprocedural Optimizations

Profile Feedback

Affects many compiler optimizations

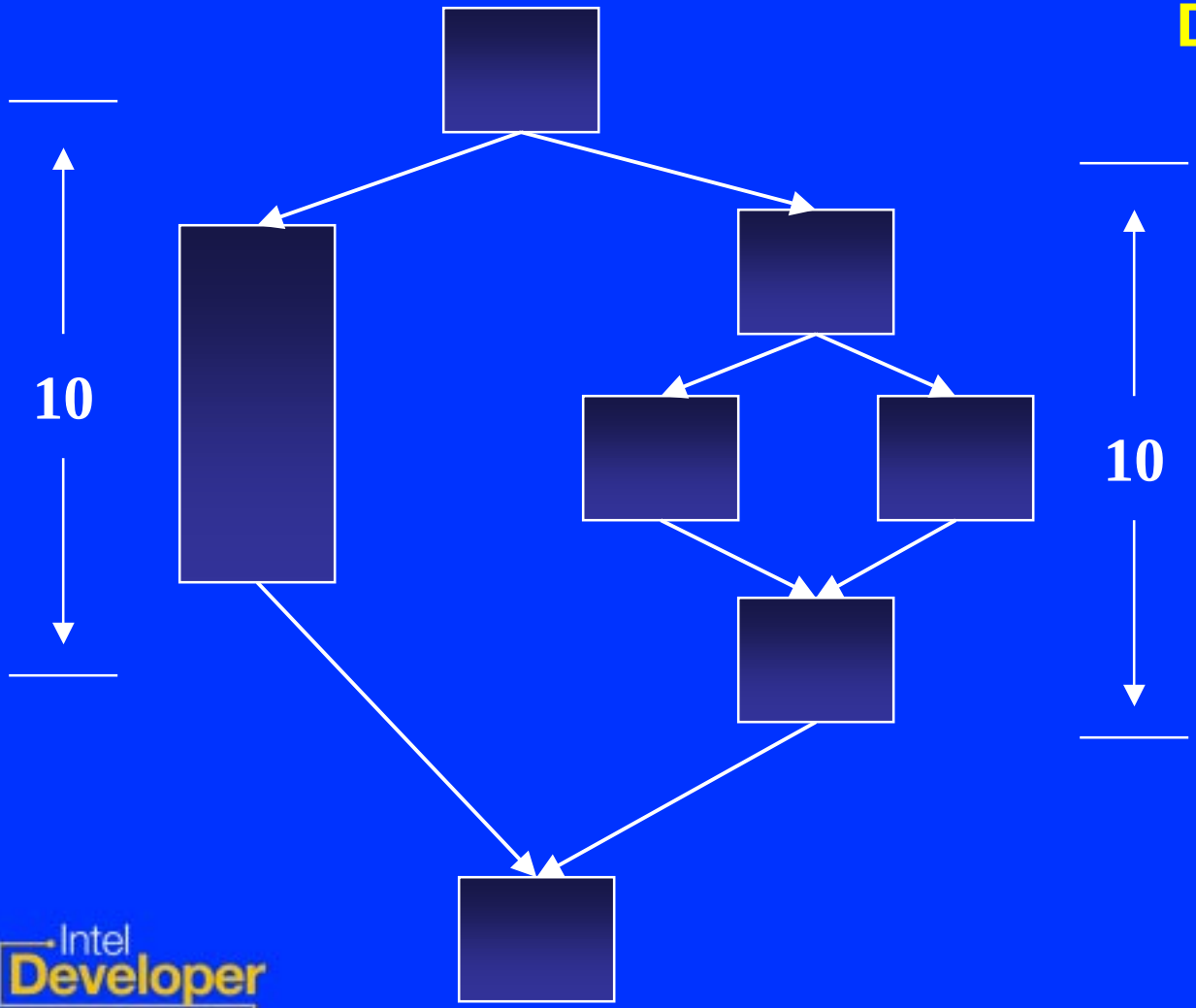
- Predication
- Speculation



IA-64 specific

- Loop (Pipeline vs. unroll vs. nothing)
- Classical (inlining, reg alloc, block order)
- Function Splitting

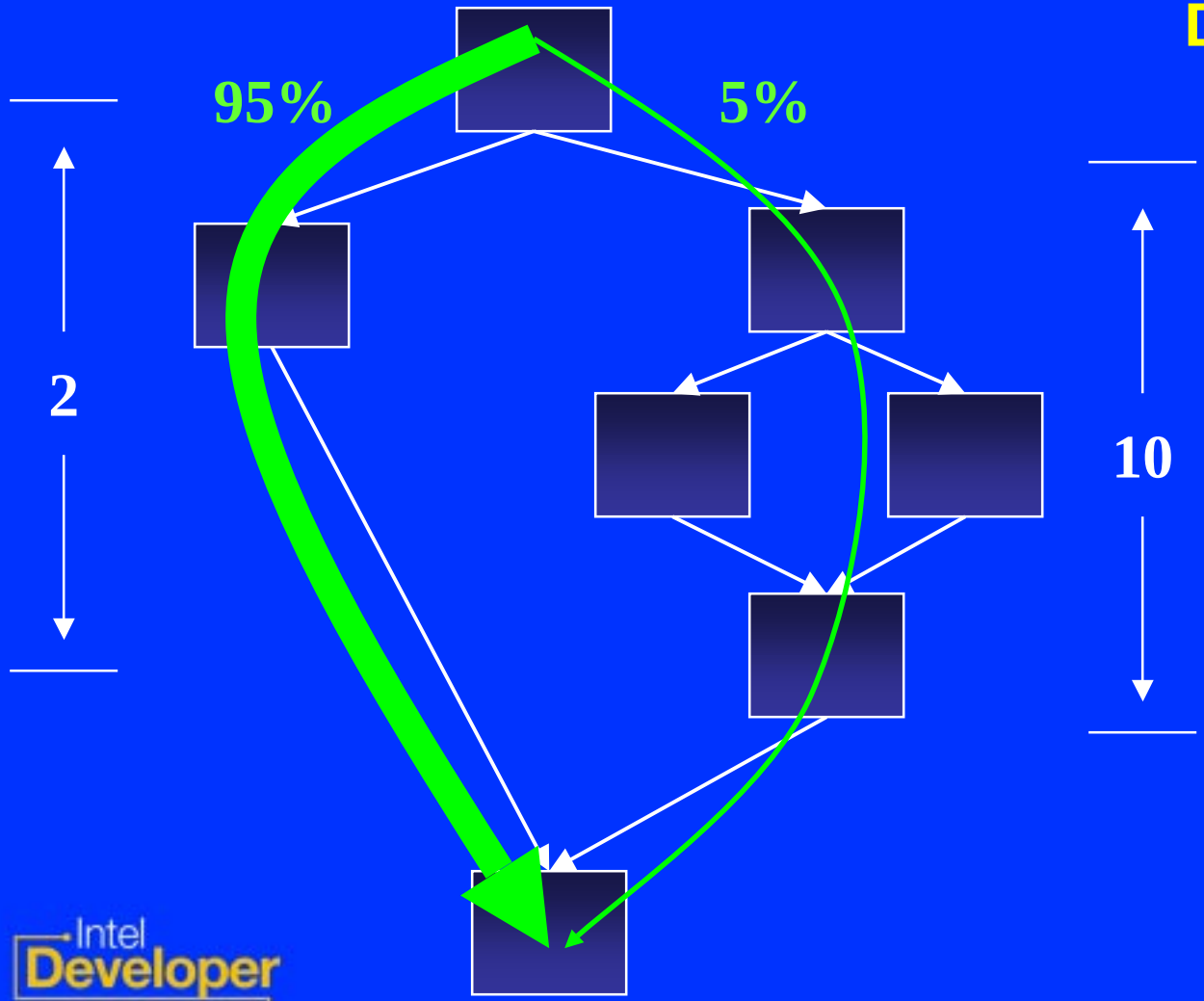
Predication



Do we predicate?

Balanced Paths - Good opportunity to predicate independent of profile

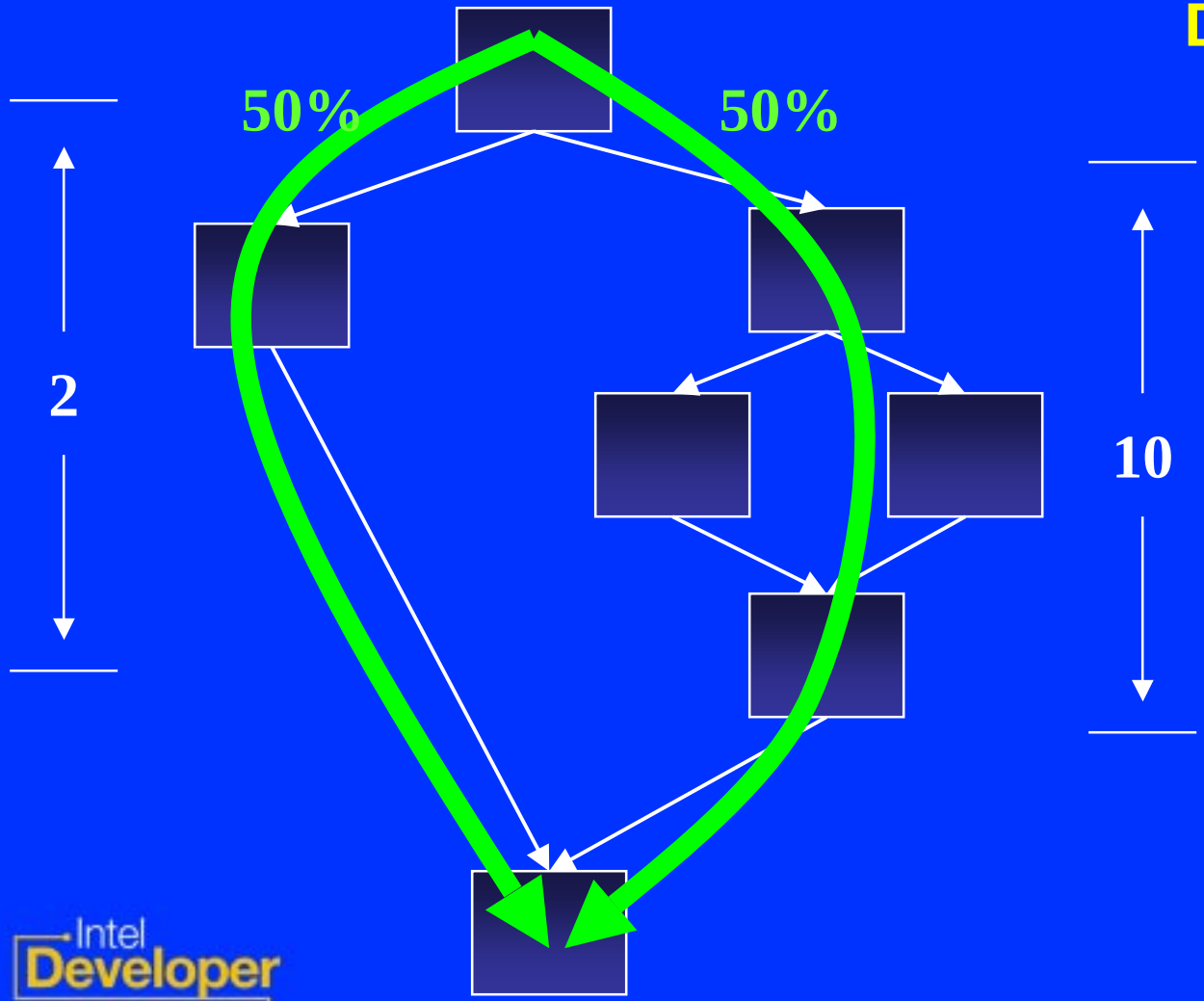
Predication



Do we predicate?

Bad Move! Main path length increased from 2 to 10.

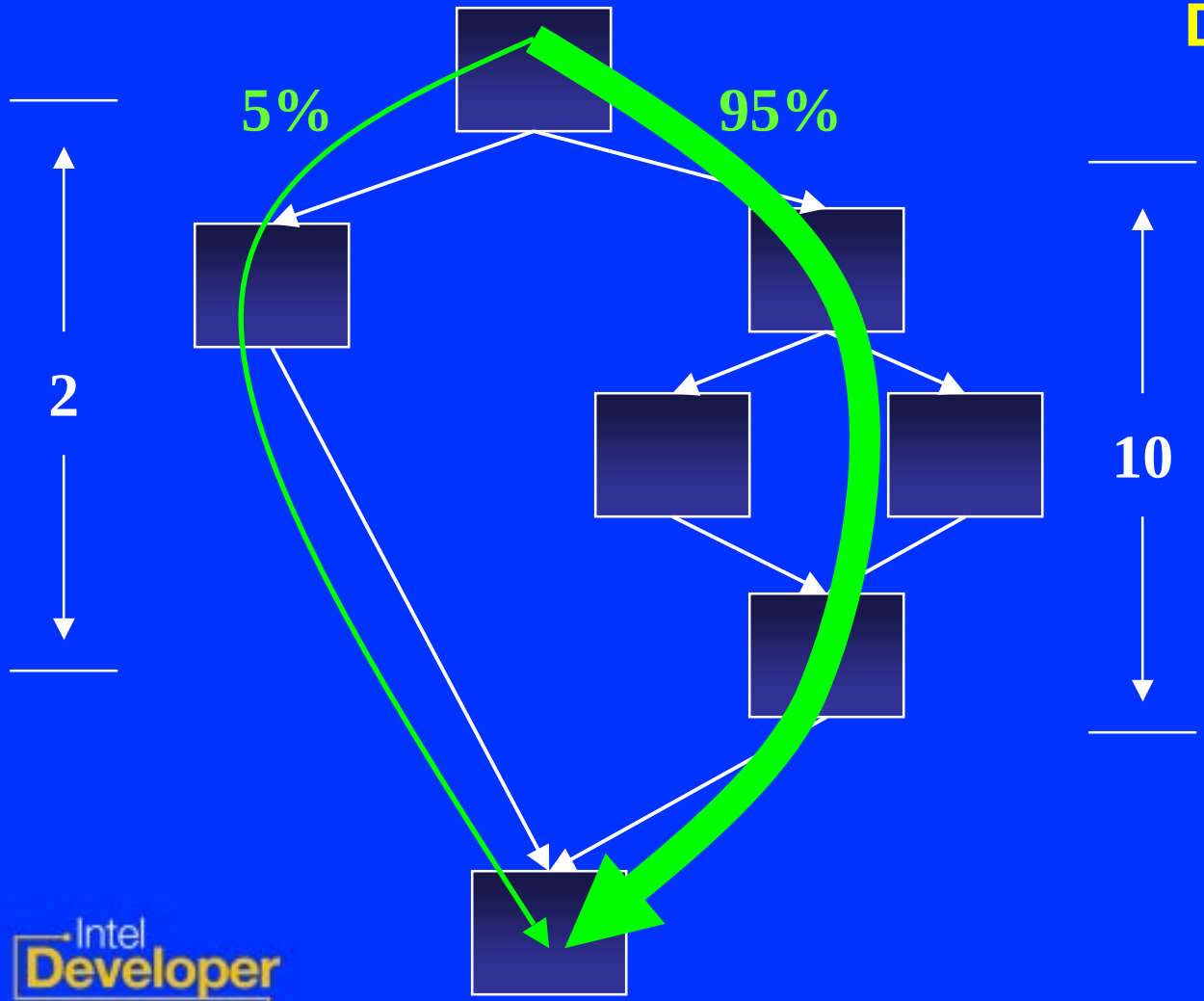
Predication



Do we predicate?

Not as clear. Main path length increased but mispredicts reduced.

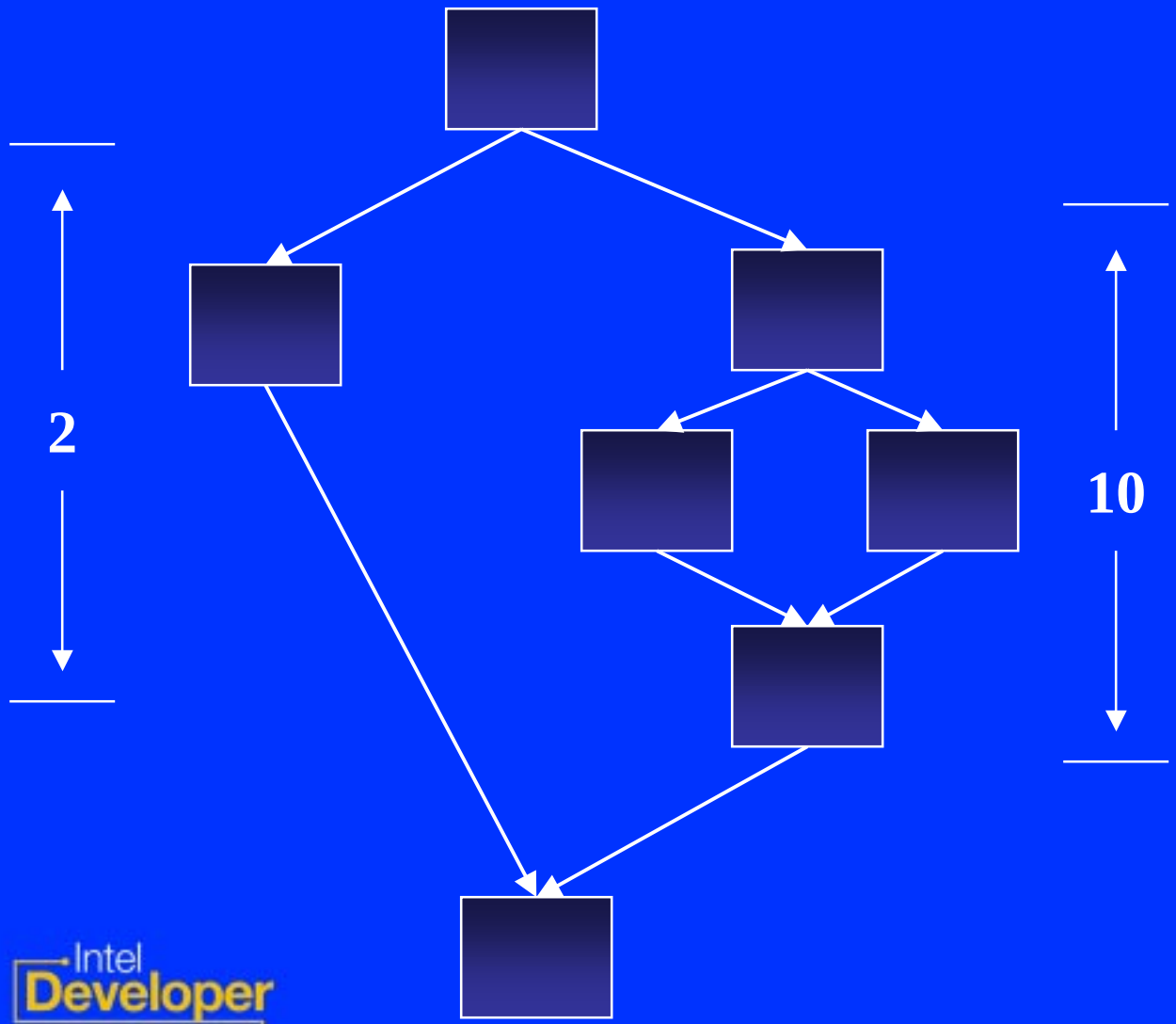
Predication



Do we predicate?

Good move. Left side will slide in for free.

Predication

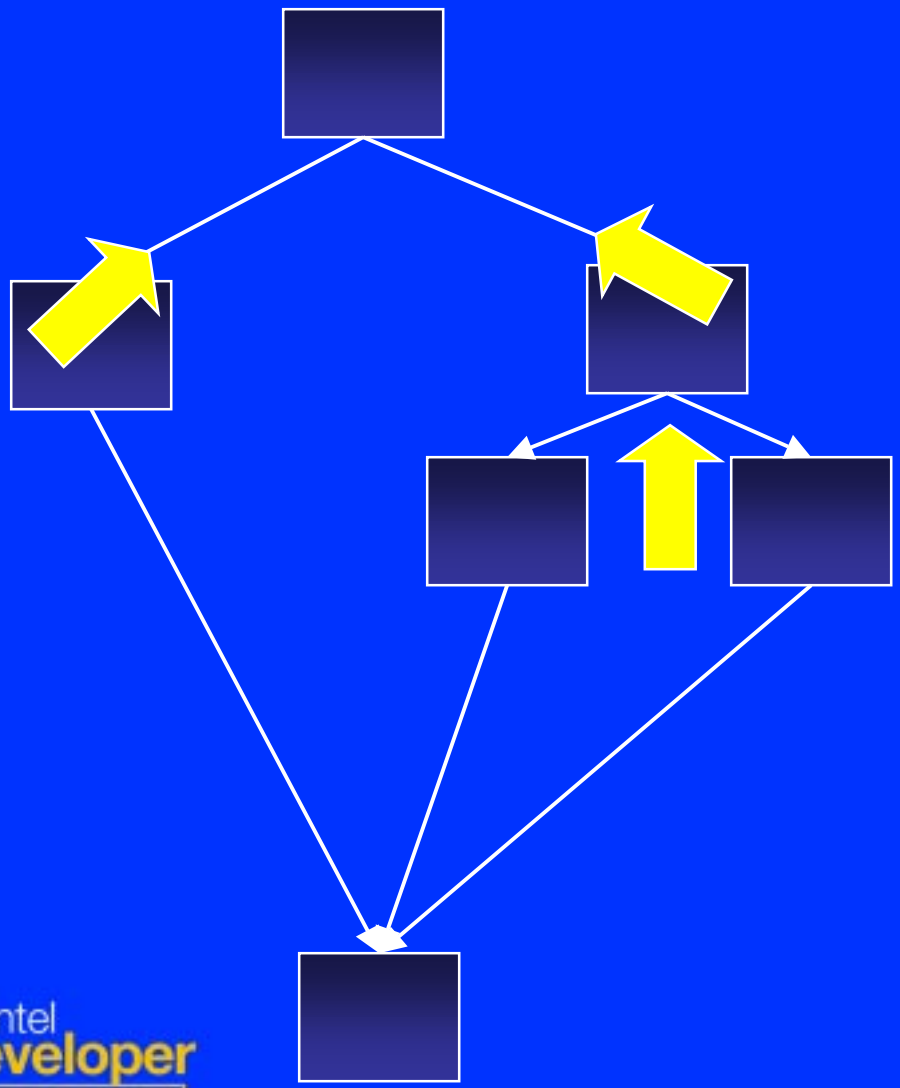


Without profile we won't predicate.

Not any worse than traditional architectures.

Forfeit chance to improve performance.

Speculation

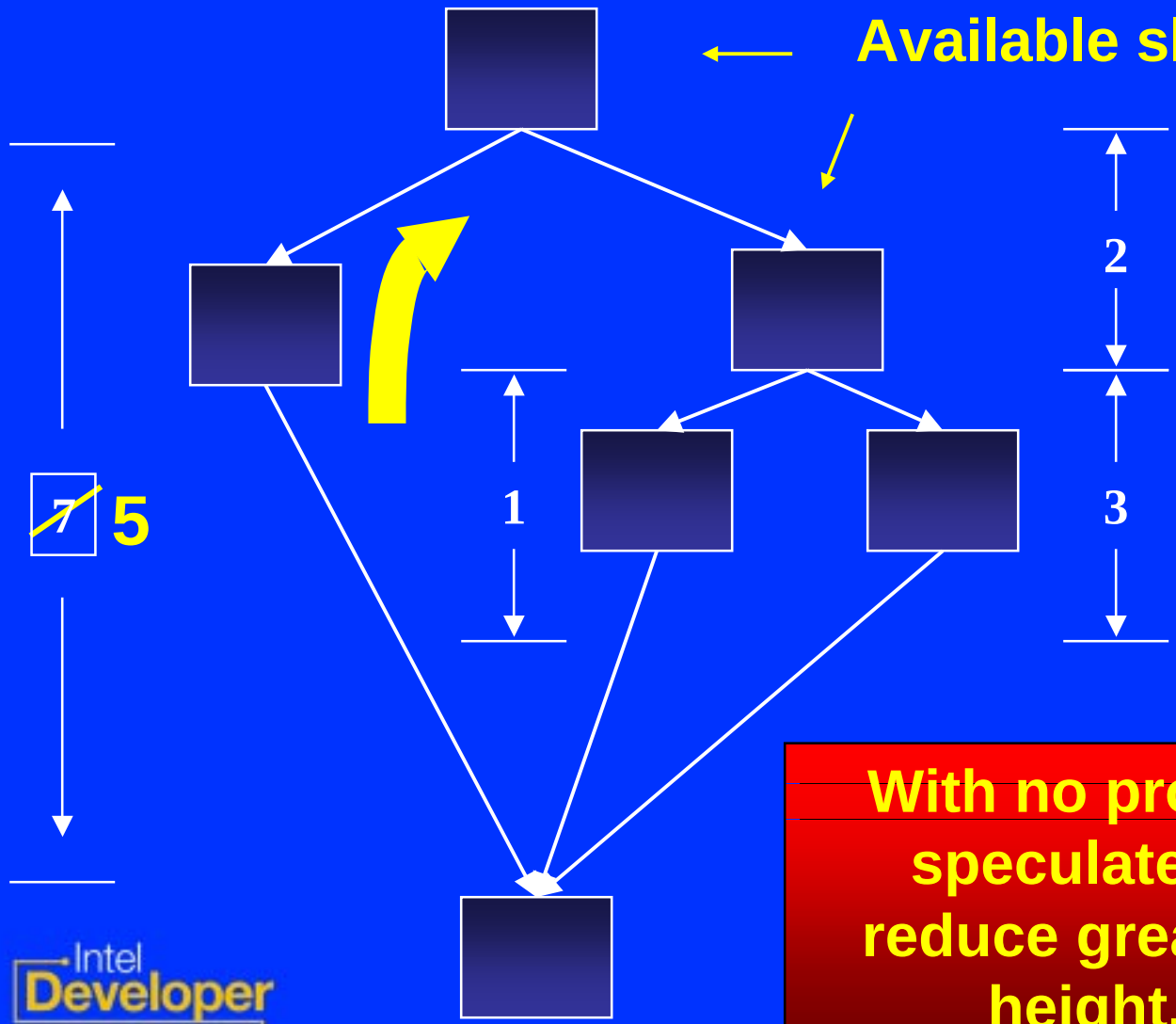


Schedule code based upon control and data dependencies.

Then speculate instructions to fill unused resources. Move code upwards.

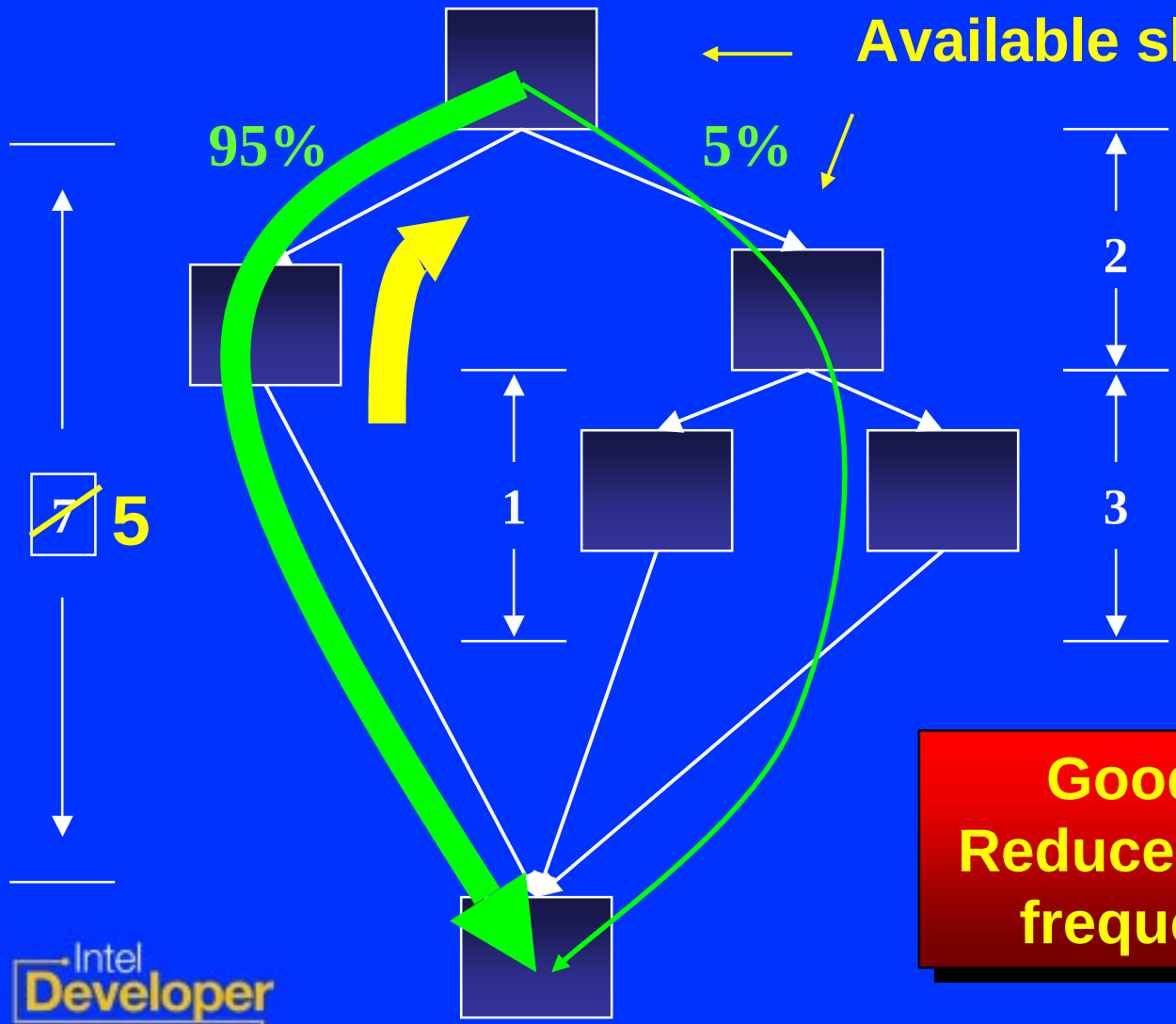
Decisions based upon path height and profile weight.

Speculation



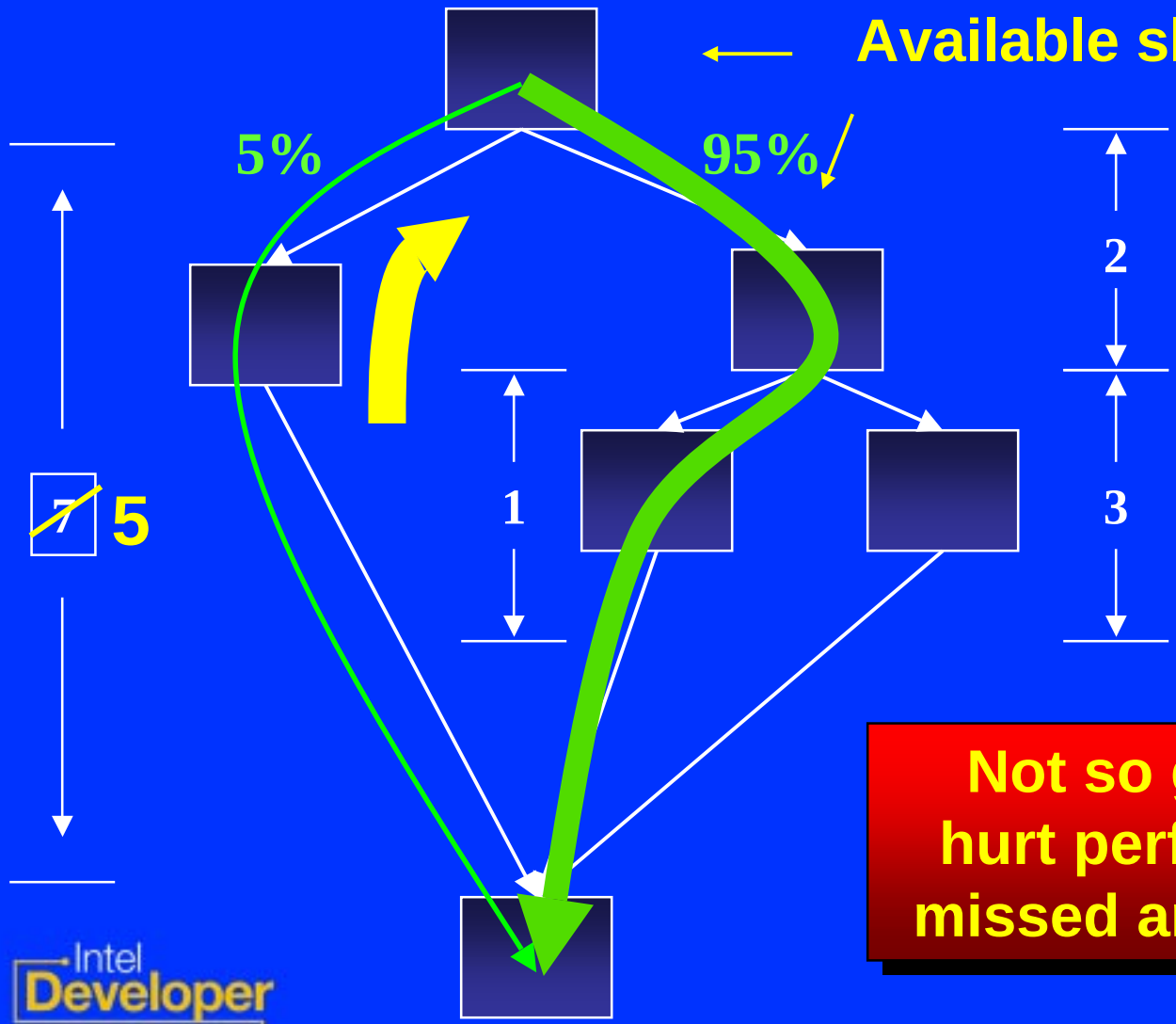
With no profile, speculate to reduce greatest height.

Speculation



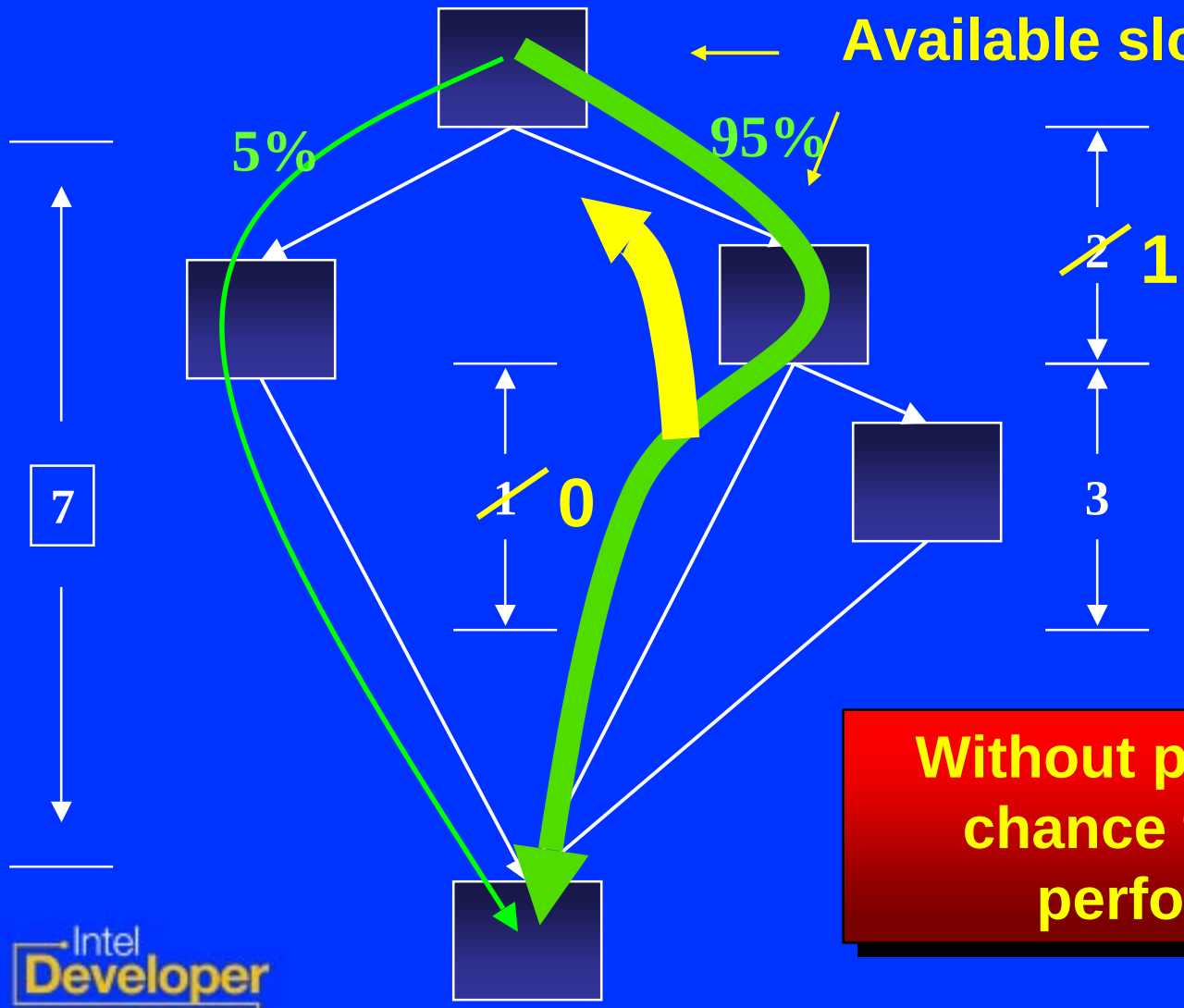
Good move. Reduced height of frequent path.

Speculation



Not so good. Didn't hurt performance but missed an opportunity.

Speculation



More useful speculation

Without profile, forfeit chance to improve performance.

Loop Optimizations

When and by how much to unroll?

When to peel?

- Heuristic without profile is based upon
  - resource, disam, register pressure
- Additionally, with profile info
  - Trip count estimate
  - Global scope for code size, register pressure

Classical Optimizations

- Register allocation
  - Spill/fill will be placed in cold regions
- Code layout
  - Separate hot and cold regions
    - Reduces cache and page faults
    - Recovery code already separated
- Function inlining
  - Hot functions will be inlined
  - Partial inline hot parts of function

Partial Inline Example

```
...
foo (p)
...
```


foo

```
if (p==NULL)
    return;
else {
   p->value = 10;
 …
}
```

Function foo has early return

Partial Inline Example

```
...

if (p!=NULL)  {
   call foo2 (p)
}
...
```

foo2

```
p->value = 10;
 …
```

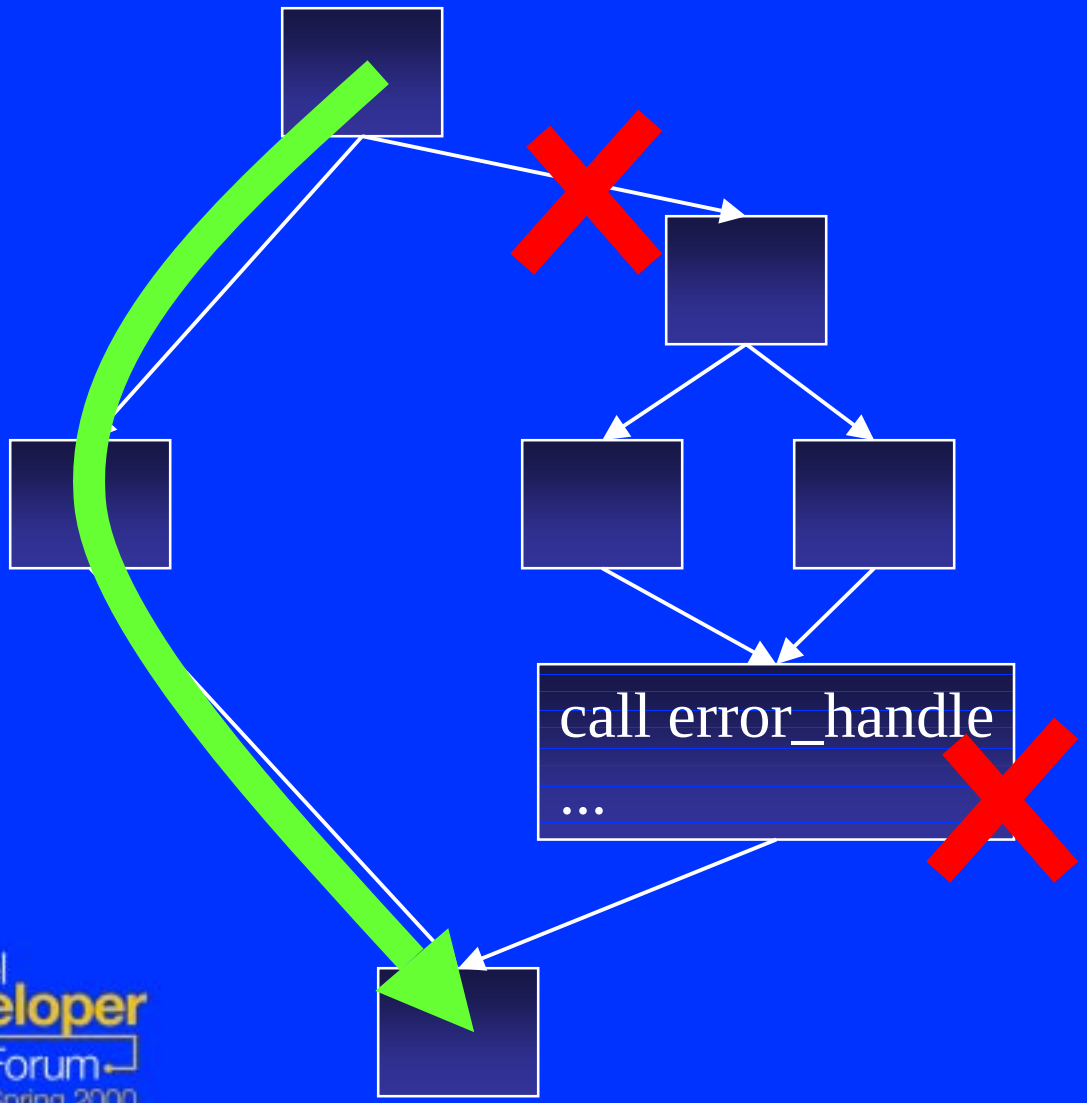
If early exit is frequent, move condition into caller

Remove overhead of call

Absence of Profile

- Static heuristics
  - Loop-back branches taken
  - Equality compares are unlikely
  - Case statements - most likely first
  - Function keywords (error or exit)
  - Compiler will propagate info to larger regions

Function Naming



Identify rarely taken function

Propagate upward

Assign profile weights

Not removing blocks!

Only guessing the profile.

Absence of Profile

- Separate hot/cold regions
  - Reduces cache and page affects
  - Fine grain within the procedure
  - Course grain during linking
- Use the ___inline keyword

Profiling message

Helps all over the place

Will give big win in IA-64

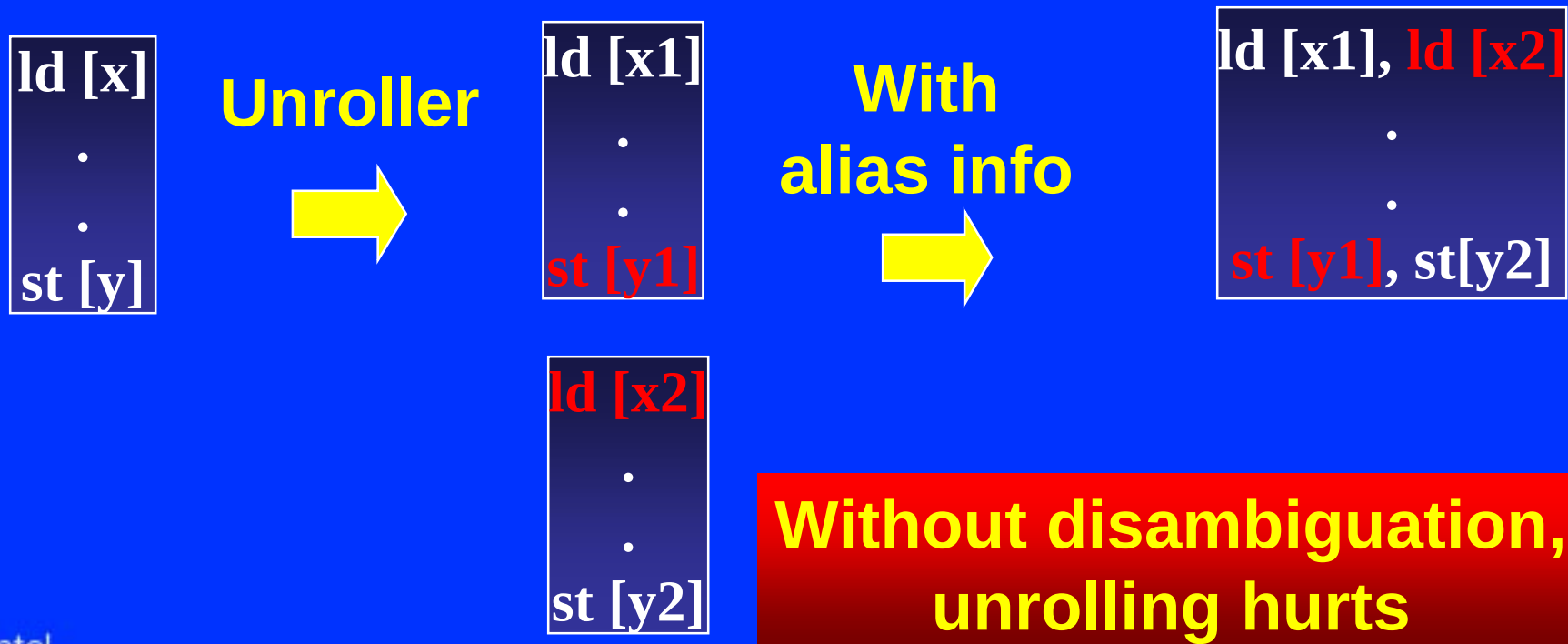
Use it!

Agenda

- Good right out of the box
- How you can make it even better
  - Profile Feedback
  - Alias Analysis
  - Interprocedural Analysis

Motivation

Benefit to unrolling and SWP

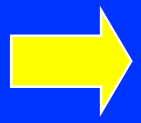


Without disambiguation, unrolling hurts

Motivation

Calls also restrict movement

```
x = status;
foo();
ptr = ptr +4;
```



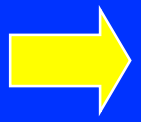
```
r1=ld [status]   ___   call foo
r2=ld [ptr]      ___   ___
r2=add r2, 4     ___   ___
```

Without disambiguation, code motion limited

Motivation

Figure out that nothing in foo modifies ptr

```
x = status;
foo();
ptr = ptr +4;
```



```
r1=ld [status] r2=ld [ptr] call foo
r2=add r2, 4    ___    ___
```

Alias info is key to many optimizations

Analysis Assistance

- When using pointers, use them responsibly (common programming practices)
- Good compilers can overcome many problems but ...
  - Compilers vary in capabilities
  - Optimization phase ordering

Responsible Pointer Use

- Use array-based addresses when possible
- Follow ANSI rules
  - Allows type-based disambiguation
- Point to beginning rather than middle of structures
- Direct references always better than through a pointer

More things to avoid

- Don’t take address of variable if not necessary
- Don’t point to same memory address with 2 different pointers
- Avoid pointer arithmetic

Analysis Assistance

- Command line options
  - Oa and Ow command-line options
  - Qansi command-line option
- Keywords and pragmas
  - *Optimize* pragma
  - *Restrict* keyword
  - *Unlikely_alias* keyword

Command line info

```
ecl -Oa file.c
```

- All pointers in file.c are unambiguous
- Sledgehammer approach

```
ecl -Ow file.c
```

- All pointers in file.c are unambiguous wrt each other but not wrt to function calls

```
ecl -Qansi file.c
```

- Compiler can assume type-based memory disambiguation

Back to the saxpy loop

```
void foo(int *x, int *y, int *z) {
    int i;
    for (i = 0; i < 100; i++)
        z[i] = A * x[i] + y[i];
}
```

- Recall 4 cycles/iteration with SWP and data speculation
- Alias information will allow 3 cycles/iteration

Pragma Example

```
#pragma optimize (“a”, on)
void foo(int *x, int *y, int *z) {
    int i;

    for (i = 0; i < 100; i++) {
     z[i] = A * x[i] + y[i];
    }
}
#pragma optimize (“a”, off)
```

Use optimize pragma to free all pointers

Keyword Example

```
void foo(int *x, int *y, int * restrict z) {
    int i;

    for (i = 0; i < 100; i++) {
     z[i] = A * x[i] + y[i];
    }
}
```

Use restrict keyword to be more selective

Keyword Example

```
void foo(
    int *x, int *y, int * unlikely_alias z) {
     int i;

     for (i = 0; i < 100; i++) {
      z[i] = A * x[i] + y[i];
     }
}
```

Use unlikely_alias if unsure or small chance - hint to data speculate

Alias Analysis Summary

Info needed all over the place

Use pointers in a friendly manner

Add source info where useful and possible

Agenda

- Good right out of the box
- How you can make it even better
  - Profile Guided Optimizations (PGO)
  - Alias Analysis
  - Interprocedural Analysis

Interprocedural Analysis

- Gain analysis information between different procedures
  - mod/ref, address taken, call sites
- Whole program IP
  - Used when compiler can see whole app
  - Turn on using -Qipo_wp
- Single file IP
  - Use in presence of DLLs
  - Turn on using -Qipo

Coding Techniques

- Function Scoping
  - Place caller and callee in same file
  - Mark functions static if possible
    - Enables mod/ref info gathering
    - Know all callsites
    - Enables full inline
    - Function specialization

Coding Techniques

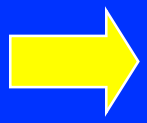
- Data Scoping
  - Local variables are best
  - Mark global variables as static whenever possible
    - Can tell if address is taken
    - Can registerize over function calls

Scoping Problem

```
int inc,
    outc;

int compress() {
   while () {

        inc++;
        call output;
        outc++;
   }
}
```



```
loop:
   ld r1 = [inc]
   add r1=r1,1
   st [inc] = r1
   call output
   ld r2 = [outc]
   add r2=r2,1
   st [outc] = r2
   branch loop
```

Compiler must keep values in memory

Scoping Problem Resolved

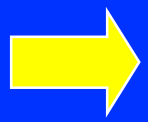
IP

```
static int inc,
     outc;

int compress() {
    while () {

          inc++;
          call output;
          outc++;

    }
}
```



```
ld r1 = [inc]
ld r2 = [outc]
loop:
    add r1=r1,1
    add r2=r2,1
    call output
    branch loop
st [inc] = r1
st [outc] = r2
```

Compiler can do its stuff

IP Summary

Use whole program IP if possible

Otherwise think single file scope when coding

Hand Optimizations

- Unnecessary and often detrimental
  - Makes things more difficult for the compiler
  - Renders source uarch-specific
- Examples
  - Don’t hand unroll loops
  - Don’t destroy perfectly nested loops
  - Try compiler-generated prefetch first

Summary

- Compiler will do the work
- Enable the compiler to do its job better
  - Use profile feedback
  - Make alias analysis easier
  - Code with interprocedural analysis in mind

Collaterals

- **IA-64 Technical information**
  http://developer.intel.com/design/ia-64/architecture.htm
- **IA-64 Computer Based Tutorials**
  http://developer.intel.com/vtune/cbts/ia64/index.htm


Intel
Developer
Forum
Spring 2000
intel


Intel
Developer
Forum
Spring 2000
intel