



Optimizing Software With Intel Compilers

By Eric W Moore



Objectives

- After completing this class, you will be able to:
 - Optimize software for Intel® architecture
 - Use key compiler optimization switches

Agenda

- Introduction
- Optimization with switches
 - General and processor-specific optimization
 - Interprocedural optimization (IPO)
 - Profile-guided optimization (PGO)
- Compiler Reports

EPIC Architecture Principles

- “The compiler should play the key role in designing the plan of execution, and the architecture should provide the requisite support for it to do so successfully.
- The architecture should provide features that assist the compiler in exploiting statistical ILP.
- The architecture should provide mechanisms to communicate the compiler’s plan of execution to the hardware.”
 - Schlansker, Michael S. and Rau, B. Ramakrishna (HP Labs),
 - “EPIC: Explicitly Parallel Instruction Computing”
 - Computer, Vol 33 Issue: 2 , Feb. 2000 pp 37-45

Agenda

- Introduction
- Optimization with switches
 - General and processor-specific optimization
 - Interprocedural optimization (IPO)
 - Profile-guided optimization (PGO)
- Compiler Reports

General Optimization Switches

	Linux*
Disable optimization	-O0
Optimize for speed without increasing code size	-O1
Optimize for speed (default)	-O2
High-Level optimizer	-O3
Create symbols for debugging	-g
Generate assembly files with .s and stop the compilation process	-S

Intel, Itanium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.

High-level Optimizer

- Overview
 - Loop level optimizations (including loop unrolling, cache blocking, etc.)
 - Converts source code algorithm to use more optimal memory access pattern
- How
 - (Linux*) -O3
 - (Windows*) /O3
- Loops must meet certain criteria ...

High Level Optimizer

- Single precision floating point code fragment

```
for (j=1; j<1000; j++) {  
    y[j] = y[j] + a*x[j]; }  
}
```

- turns into:

```
for (j=1; j<1000; j+=2) {  
    y[j] = y[j] + a*x[j];  
    y[j+1] = y[j+1] + a*x[j+1];  
}
```

- Now it can use `ldfp` instead of `ldf`

OpenMP* Support

- OpenMP 2.0 for Fortran and C++
 - The OpenMP* 2.0 WORKSHARE directive is now supported in Fortran9.0.
- Also supports the OpenMP extension “workqueuing model” from Intel to exploit task level parallelism.
- Usage Model:

Linux*

-openmp

-openmp_report[n]

Intel, Itanium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.

Auto-parallelization

- Auto-parallelization: the automatic threading of loops without having to manually insert the OpenMP* directive
 - Compiler can identify “easy” candidates for parallelization
 - (Linux*) -parallel
 - OpenMP directives give greater flexibility

Agenda

- Introduction
- Optimization with switches
 - General and processor-specific optimization
 - Profile-guided optimization (PGO) Interprocedural optimization (IPO)
- Compiler Reports

Profile-Guided Optimizations (PGO)

- Uses execution-time feedback to guide optimization
- Helps I-cache, paging, branch prediction
- Enabled optimizations:
 - Basic block ordering
 - Better register allocation
 - Better decision of functions to inline
 - Function ordering
 - Switch-statement optimization

Usage: Three-Step Process

Step 1

Instrumented Compilation
(Linux*) `icc -prof_gen prog.c`

**Instrumented
executable**

Step 2

Instrumented Execution
Run program on a typical dataset

**DYN file containing
dynamic info: .dyn**

Step 3

Feedback Compilation
(Linux) `icc -prof_use prog.c`

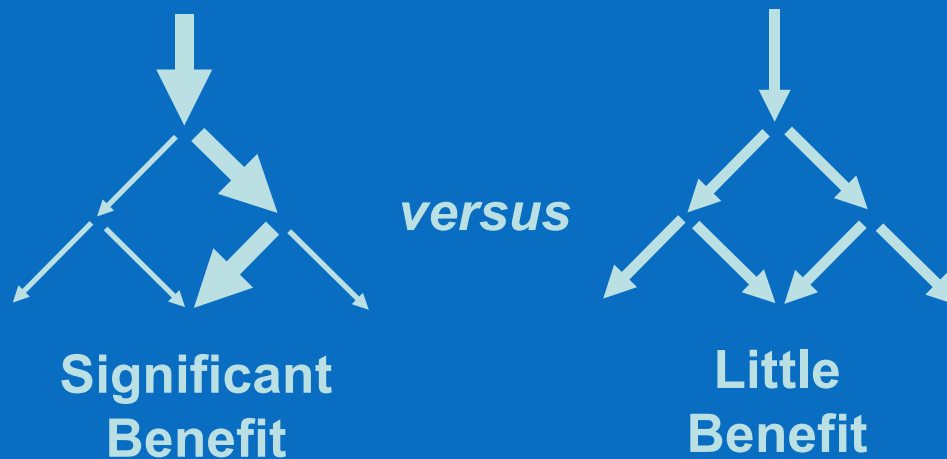
**Merged DYN
summary file: .dpi**
Delete old dyn files if
you do not want the info
included

When to Use

- Applications with lots of functions, calls, or branches
 - Examples: databases, decision-support (enterprise), MCAD
 - Applications with computation spread throughout; not confined to kernels
 - Trip counts for loops can enable more aggressive optimization
- Considerations:
 - Different paradigm for builds - three steps
 - Schedule time in final stages of development when code is more stable
 - Use representative data sets (not for corner cases)

Programs That Benefit

- Consistent hot paths
- Many if statements or switches
- Nested if statements or switches

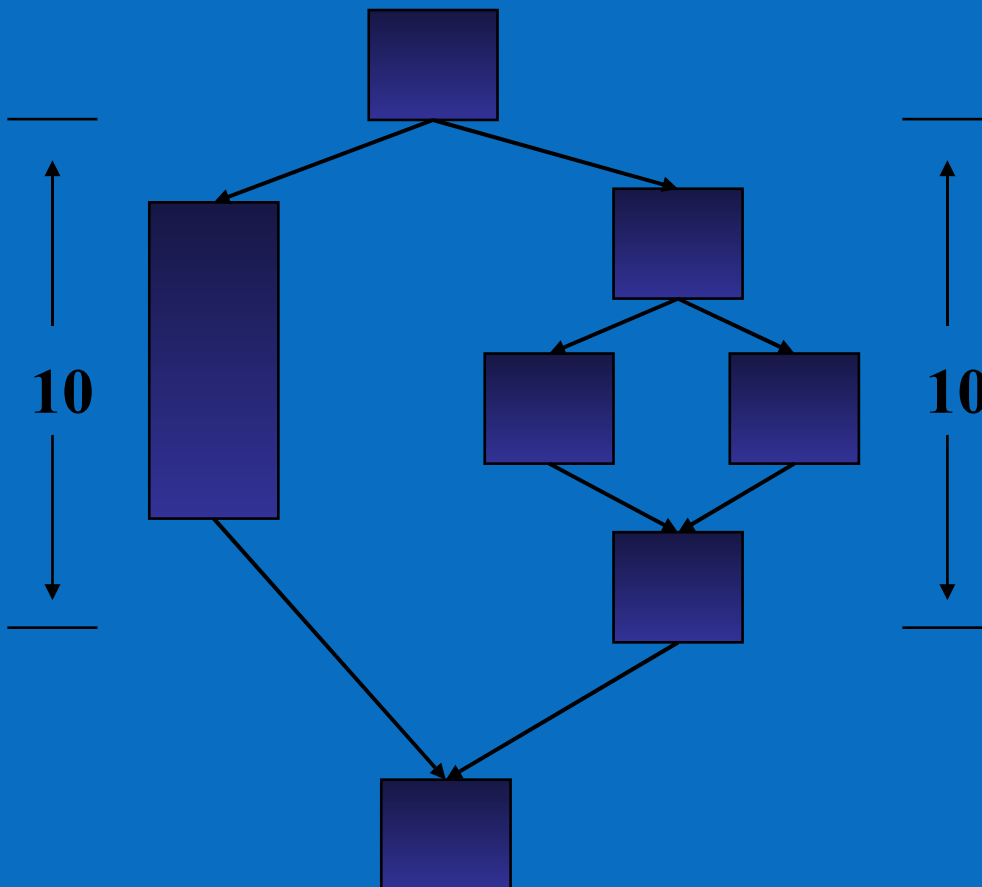


Intel, Itanium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.

Profile-Guided Optimizer

- Affects many compiler optimizations
 - Dynamic branch prediction
 - Loop (pipeline versus unroll versus none)
 - Cache utilization
 - Predication
 - Speculation
 - Classical (block order, inline, reg alloc)
 - Function splitting

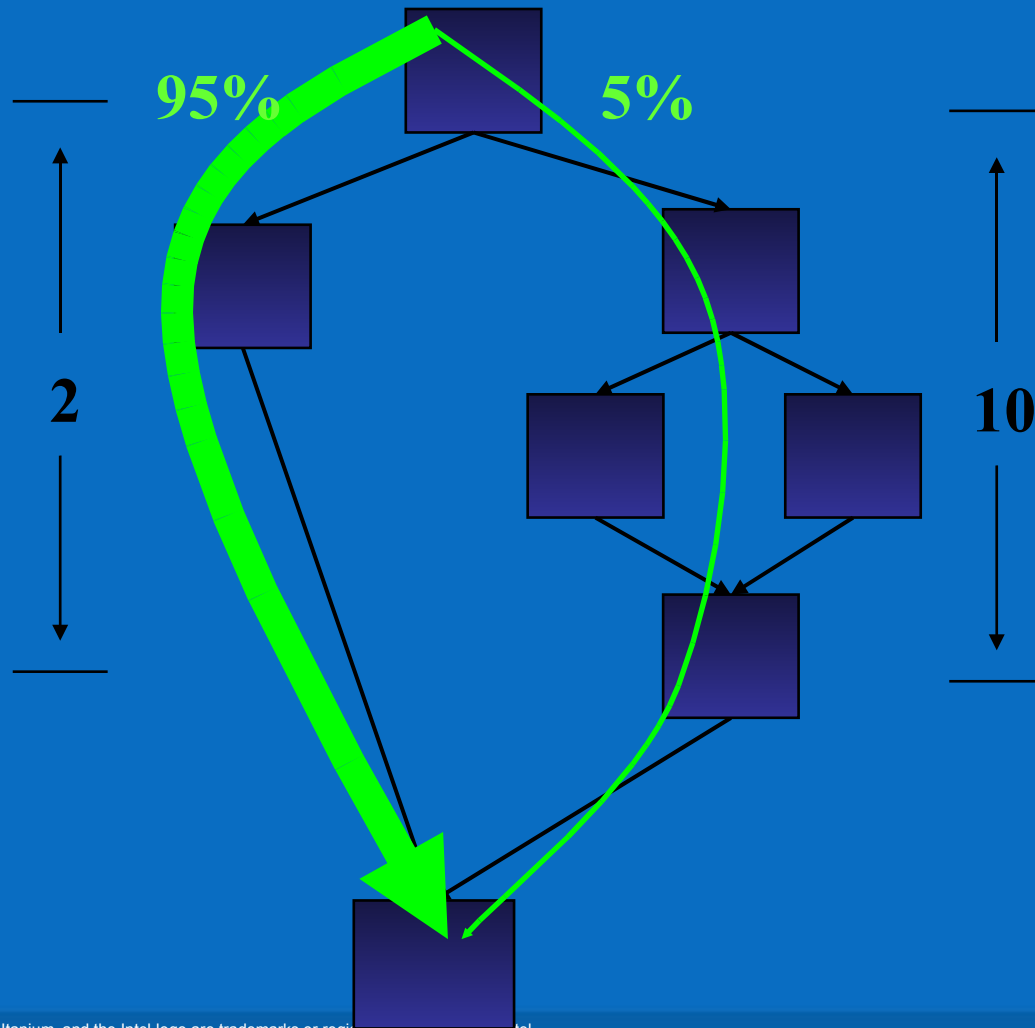
Helps Improve Predication



Do we predicate?

**Balanced Paths:
Good opportunity
to predicate
independent
of profile**

Predication: PGO Benefits

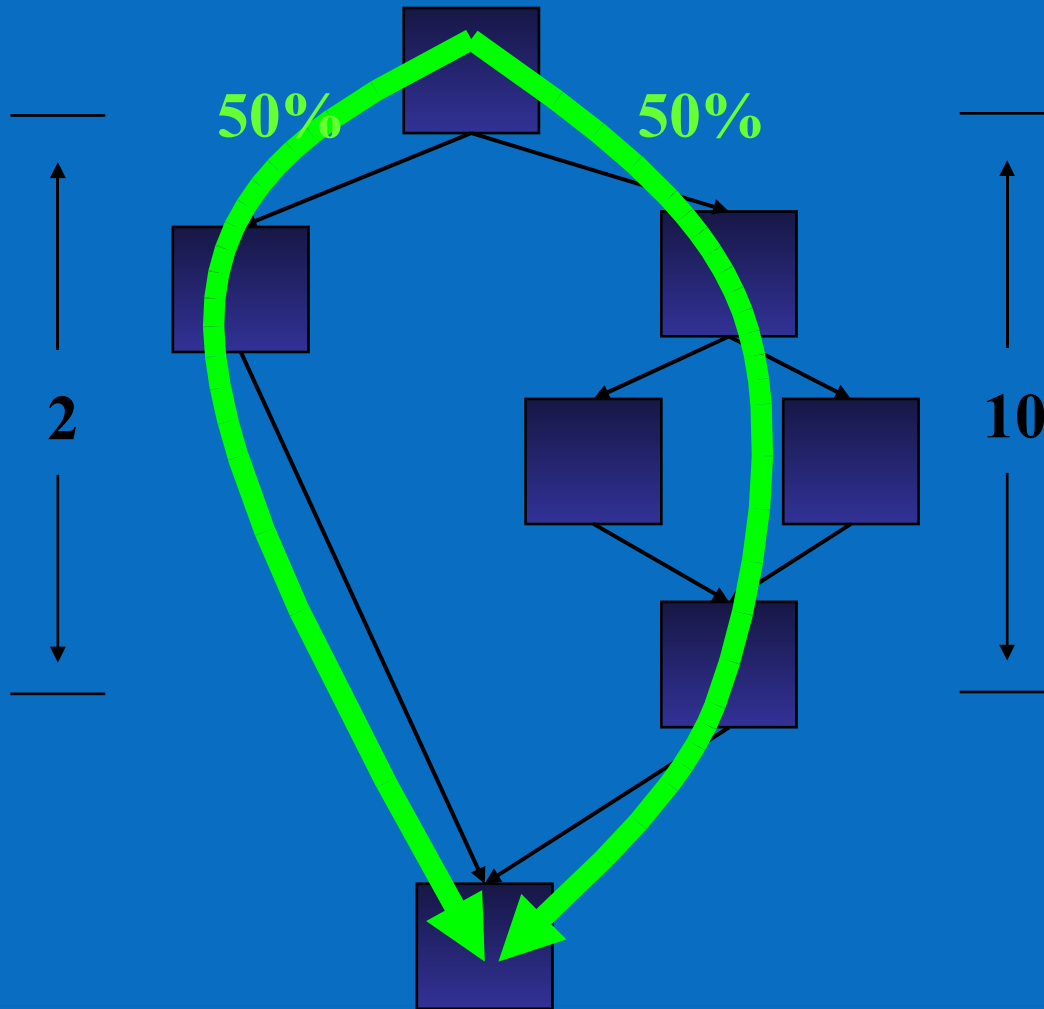


Do we predicate?

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

Intel, Itanium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.

Predication: PGO Benefits

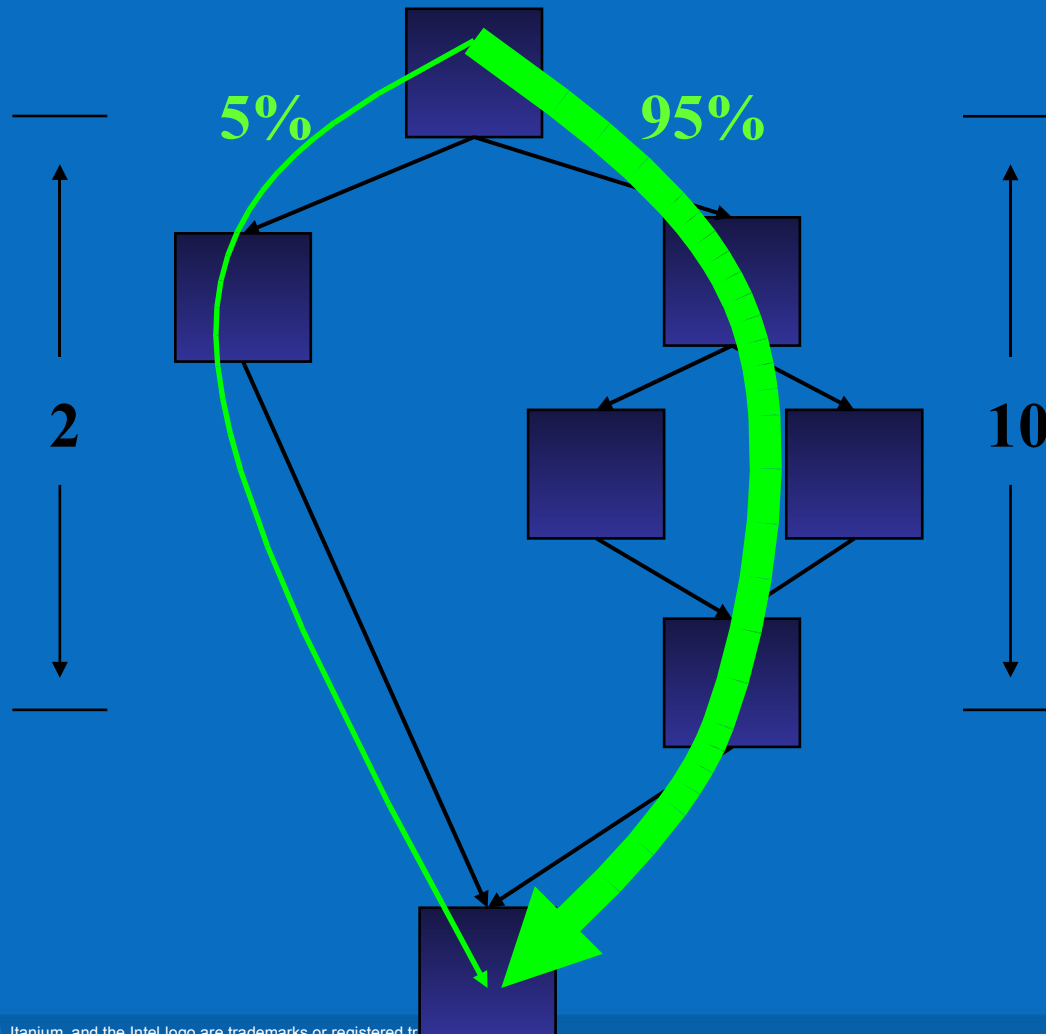


Do we predicate?

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

Intel, Itanium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.

Predication: PGO Benefits

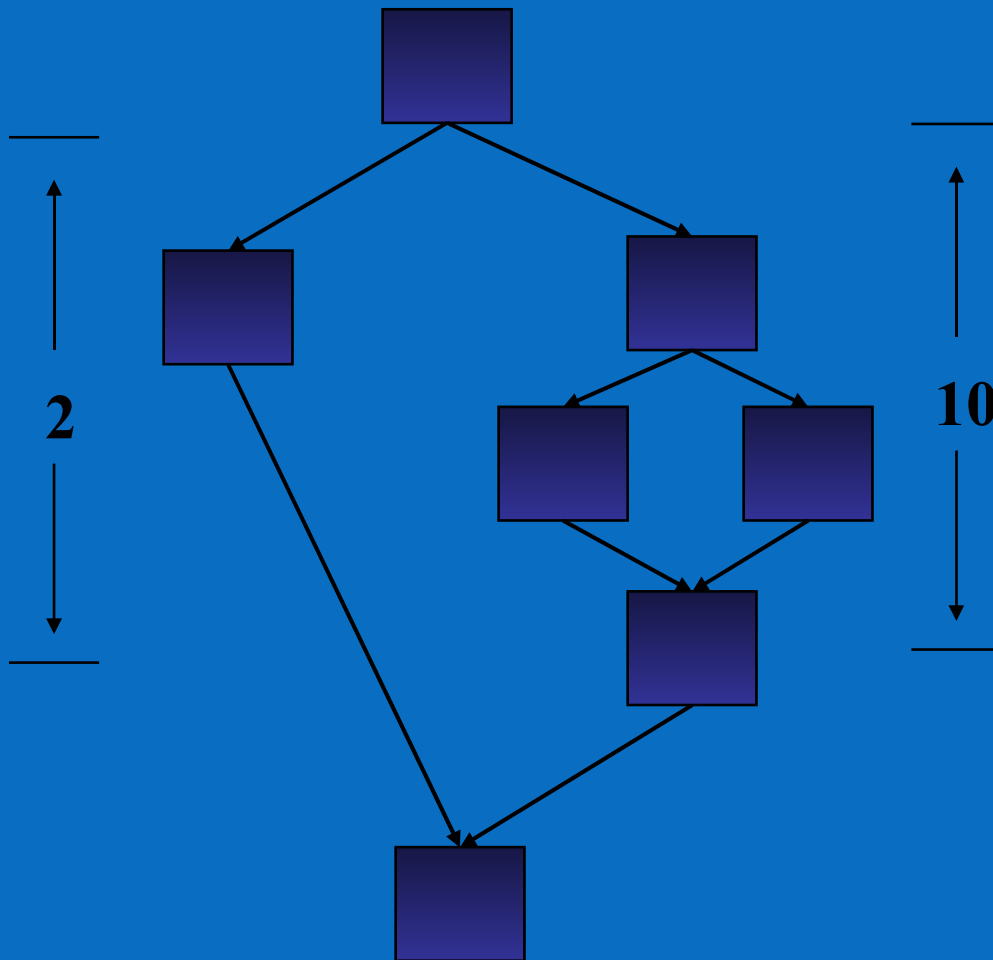


Do we predicate?

Good move.
The left side will
slide in for free

Intel, Itanium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.

Predication: PGO Benefits



**Without profile
we will not predicate**

**Not any worse
than traditional
architectures.**

**Forfeit chance to
improve
performance.**

Agenda

- Introduction
- Optimization with switches
 - General and processor-specific optimization
 - Profile-guided optimization (PGO)
 - **Interprocedural optimization (IPO)**
- Compiler Reports

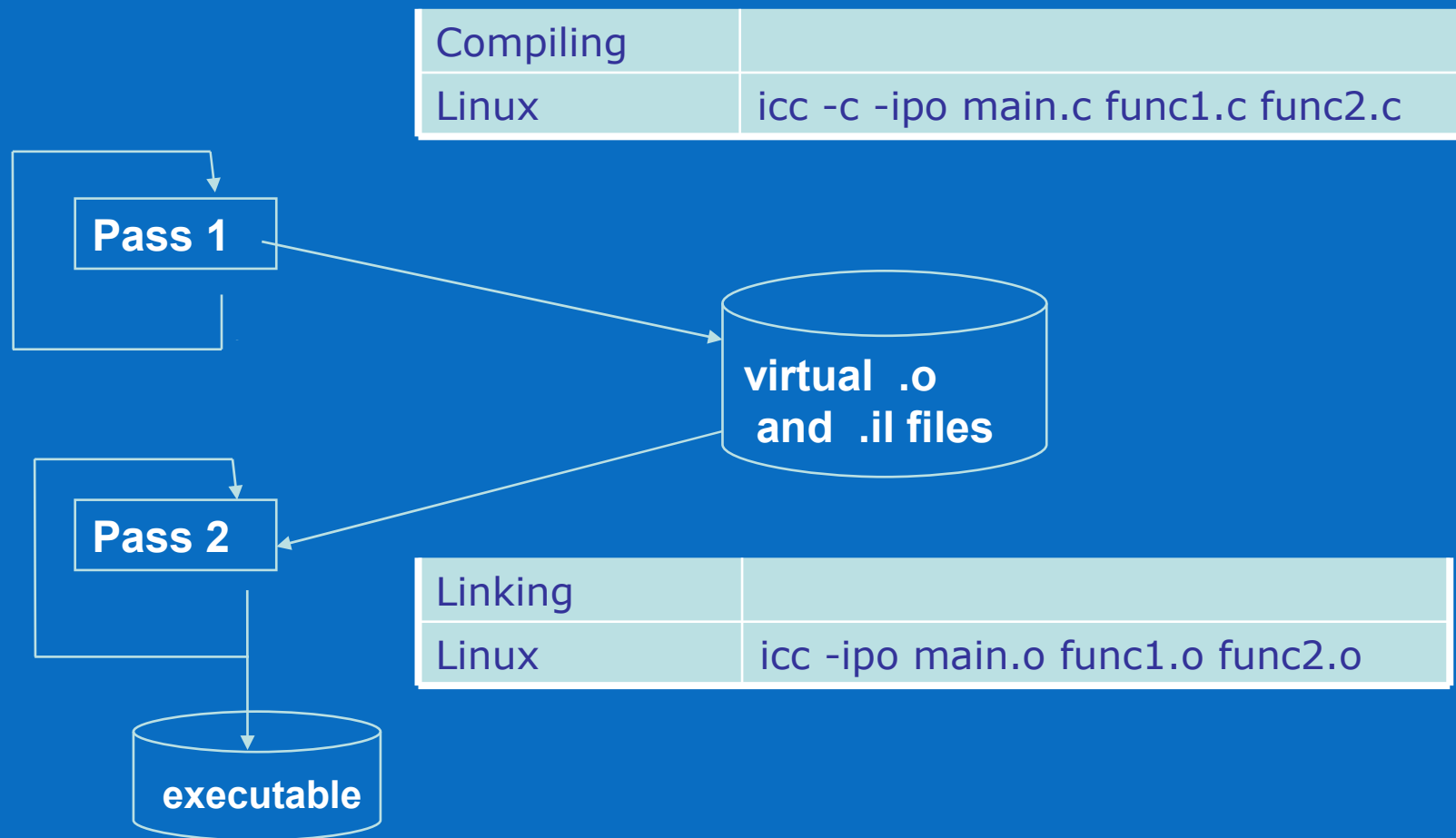
Interprocedural Optimizations (IPO)

- (Linux*) -ip
- Enables interprocedural optimizations for single file compilation.
- (Linux*) -ipo
- Enables interprocedural optimizations across files
- Enhances optimization when used in combination with other compiler features

Interprocedural Optimizations (IPO)

- Benefits
 - Inlining
 - Partial inlining
 - Help in software pipelining through memory disambiguation
 - Interprocedural constant propagation
 - Passing of arguments in registers
 - Loop-invariant code motion
 - Dead code elimination

Usage: Two-Step Process



Intel, Itanium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.

Agenda

- Introduction
- Optimization with switches
 - General and processor-specific optimization
 - Interprocedural optimization (IPO)
 - Profile-guided optimization (PGO)
- **Compiler Reports**

Generating Compiler reports

- Generating the full report
 - Compilers 9.1 and earlier
 - `icl [optimizations] /Qopt_report /Qopt_report_level max /Qpar_report3 &> compreport.txt`
- Note: The compiler will only generate a report for phases you turned on with optimization switches (i.e. no vector report generated without a vectorization switch: `-xT`)
- BKM – use `/Qopt_report_level max` - The Default level – does not identify compiler problems or identify potential solutions

Filter The Data in the Reports

- Choose only specific phases relevant to what you are looking for
 - `-opt_report_phase [phase]`
 - Enables the report for only the selected phases
 - Most Useful phases
 - `hlo`
 - `ipo_inl`
 - `ecg_swp`
 - `-par_report[0..3]`
 - Explains which loops Parallelized and why not
 - (including alias information)
- `-opt_report_routine functionname`
 - Views only the data for a particular function

Example of Inlining Report (9.1 Compiler)

```
int main(int argc, int argv[]) {  
    init(a, 1000);  
    mysum = sum(a, 1000, 3);  
    printf("%d\n", mysum);  
}
```

```
void init(int a[], int n) {  
    for (i = 0; i < 1000; i++)  
        a[i] = 0;  
}
```

```
int sum(int a[], int n, int b) {  
    int mysum = 0;  
    for (i = 0 ; i < n; i++)  
        mysum = add(mysum,a[i]);  
    return mysum;  
}
```

- INLINING REPORT: (main)
- -> printf(EXTERN)
- -> INLINE: sum(5) (isz = 18) (sz = 30 (16+14))
- -> INLINE: add(6) (isz = 0) (sz = 8 (3+5))
- -> init(2) (isz = 14) (sz = 22 (14+8))
- [[Callee not marked with inlining directive or pragma]]

HLO Report – daxpy.c

High Level Optimizer Report for: daxpy

Total #of lines prefetched in daxpy for loop in line

of dynamic_mapped_array prefetches in daxpy for loop in line 5=2, dist=53

.....

Loop dual-path'd for swp:

Loop at 5 -- **selected for multiversion - DD**

**Versioned for runtime
dependence analysis**

**x, y independent
x, y not independent**

Block, Unroll, Jam Report:

(loop line numbers, unroll factors and type of transformation)

Loop at line 5(rt-dd-ver 1) unrolled with remainder by 8

Loop at line 5(rt-dd-ver 2) unrolled with remainder by 8

Loadpair Report:

Load-pair formed at line 5 , 5 [Method = Multiversion(2-way)]

...

**Versioned for
alignment**

```
void daxpy(int n, double a, double *x, double *y) {  
    int i;  for (i=0;i<n;i++) *(x+i) += *(y+i)*a ; }  
}
```

Intel, Itanium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.



New to Intel® Compiler 9.1 for HLO

Suggestions and Rationale

```
for (j=0; j<N; j++) {  
    A[j] = j + B[j][1];  
    for (i=0; i<N; i++)  
        B[i][j] += A[j];  
}
```

Loop Interchange Not Done due to: Imperfect Loop Nest

Advice: Loop Interchange, if possible, might help Loopnest at lines: 6 8

: Suggested Permutation: (1 2) --> (2 1)

Example of Par_report3

```
for(i=0;i < VAL;i++) {  
    x = h * ((double)i - 0.5);  
    p += 4.0 / (1.0 + x*x);  
}  
p*=h
```

pi.c(14) : (col. 9) remark: LOOP WAS AUTO-PARALLELIZED.

procedure: func

parallel loop: line 14

shared : { }

private : { "i" "x" }

first priv.: { "h" }

reductions : { "p" }

Software Pipeline Report Example 1

Resource II = 34
Recurrence II = 9
Minimum II = 34
Scheduled II = 57

Estimated GCS II = 160

Percent of Resource II needed by arithmetic ops = 15%
Percent of Resource II needed by memory ops = 12%
Percent of Resource II needed by floating point ops = 100%

Number of stages in the software pipeline = 3

Software Pipeline Report Example 2

Resource II = 27
Recurrence II \geq 28*
Minimum II = 27
Last attempted II = 30

Estimated GCS II = 29

*Too many cycles, swp gave up computing recurrence II

Software pipeliner estimated that it is more profitable to schedule the loop using the global acyclic scheduler than to pipeline the loop

Other Considerations

- See Compiler User's Guide and Reference
- `icc -help`

Intel, Itanium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.



Summary

- Intel® compiler major optimization switches
 - High level optimizations
 - Interprocedural optimizations
 - Profile-guided optimizations
- Intel® compiler takes advantage of current Itanium® architecture

Intel, Itanium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.



References

- Web-based and classroom training
www.intel.com/software/college
- White papers and technical notes
www.intel.com/ids
www.intel.com/software/products
- Product support resources
www.intel.com/software/products/support

Backup

Intel, Itanium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.



General and Processor-specific Switches

Instruction Scheduling for Itanium® 2 Processors

- On Itanium processors
 - The compiler must insert *blocks* of no operation (NOPS) instructions with stop bits after shift operations.
 - Use of the output of variable shifts by integer instructions (ALU, st, ld) requires a four-cycle delay or a pipeline flush (10 cycles penalty)
- On Itanium 2 processors
 - NOPs are not required on examples above
 - These operations are “scoreboarded,” removing the risk of the pipeline flush.
 - The latency for such use is 3 cycles instead of 4

General and Processor-specific Switches

Variable Shift Example

```
c = (a << (b-4));
c += a;
return c;
```

Itanium® Processor	Itanium 2 Processor
<pre>{ .mii nop.m0 sxt4 r3=r8 ;; shl r2=r32,r3 ;; }</pre>	<pre>{ .mii nop.m0 sxt4 r3=r8 ;; shl r2=r32,r3 ;; }</pre>
<pre>{ .mmi nop.m0 ;; nop.m0 nop.i 0 ;; }</pre>	<pre>{ .mib add r8=r2,r3 nop.i 0 br.ret.sptk.many b0 ;; }</pre>
<pre>add r8=r2,r32 nop.i 0 }</pre>	

Intel, Itanium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.

Miscellaneous Switches

A Few Fortran Switches...

Local Variables		
Linux*	Windows*	
-autoscalar	/Qauto_scalar	Only scalar variables are automatic (default)
-auto	/Qauto	Make all automatic (default with OpenMP*)
-save	/Qsave	Make all static
-zero	/Qzero	Initialized to zero (if static)
If you do not use Fortran 95		
-FI	/FI	Fixed format
-w95	/w95	Suppressed F95 warnings

Intel, Itanium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.

Interprocedural Optimizer

- 254.gap, integer.c, (from SPEC CPU2000)

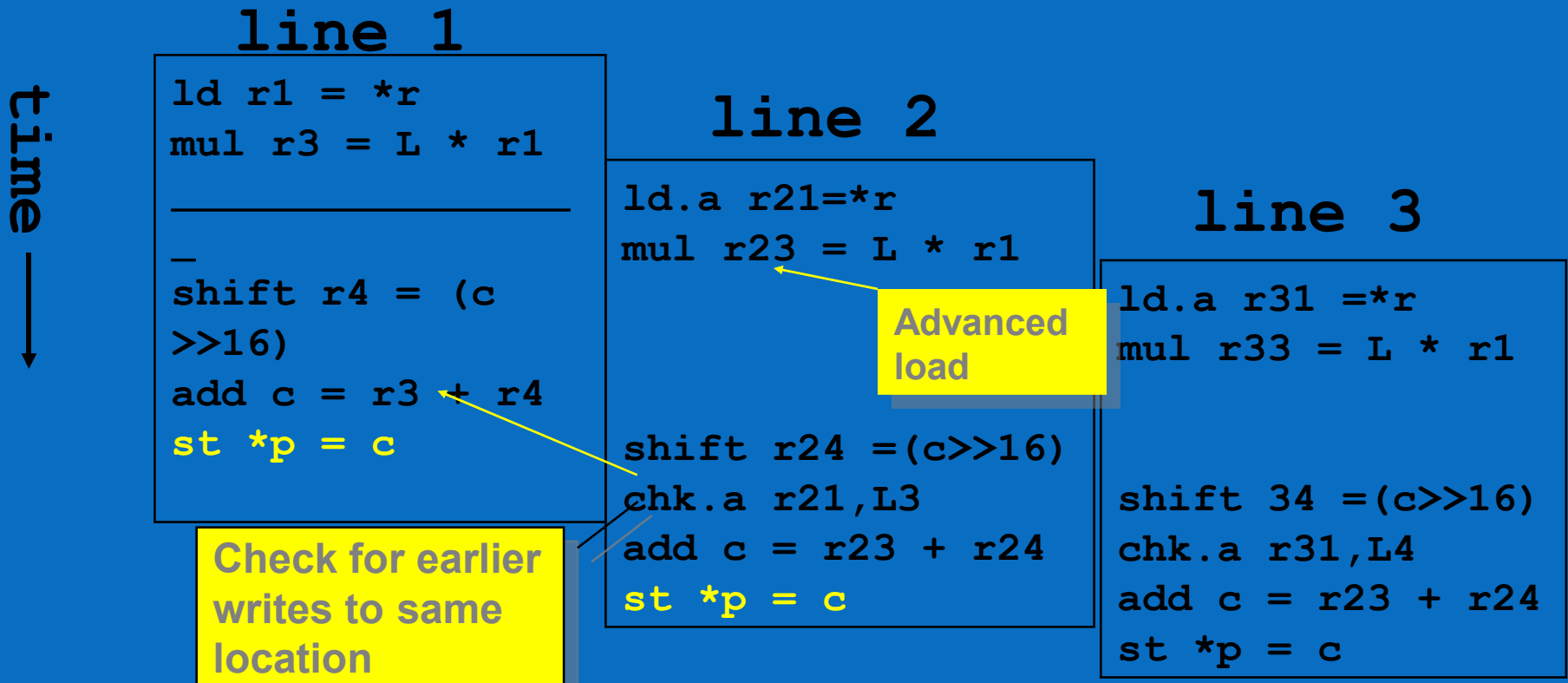
```
for((k=SIZE(hdR)/4*sizeof(TypDigit));k!= 0;--k)
{
    C = L * *r++ + (c>>16); *p++ = C; //line 1
    C = L * *r++ + (c>>16); *p++ = C; //line 2
    C = L * *r++ + (c>>16); *p++ = C; //line 3
    C = L * *r++ + (c>>16); *p++ = C; //line 4
}
```

- *r* passed in as formal parameter
- *p* is dynamically allocated

IPO can predict that *r* and *p* are independent.

Interprocedural Optimizer

- Helps memory disambiguation / data speculation
- r and p are probably independent



Miscellaneous Switches

Complex Numbers

- To use the Intel® math library, include the header file, `mathimf.h`
- Floating point Complex Example

```
#include <mathimf.h>
float _Complex c32in,c32out;
double pi_by_four= 3.141592653589793238/4.0;
c32in = 1.0 + __I__* pi_by_four;
c32out = cexpf(c32in);
printf("When z = %7.7f + %7.7f i, cexpf(z) = %7.7f + %7.7f i\n", crealf(c32in), cimagf(c32in), crealf(c32out),
      cimagf(c32out));
```

- Compile as follows:

```
icc test.c -c99
```

```
Icl test.c /Qc99
```

- If using double precision `_Complex`

```
icc test.c -c99 -long_double
```

```
Icl test.c /Qc99 /Qlong_double
```

Miscellaneous Switches

Complex Math Functions

- Each double precision function listed also has a float precision and a long double precision counterpart
 - E.g.- carg returns double precision result
 - cargf returns float precision result
 - cargl returns long double precision result

CABS	CCOSH	CONJ
CARG	CEXP	CPOW
CACOS	CEXP10	CPROJ
CACOSH	CIMAG	CREA
CASIN	CIS	CSIN
CASINH	CISD	CSINH
CATAN	CISD	CSQRT
CATANH	CLOG	CTAN
CCOS	CLOG2	CTANH

Intel, Itanium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.