Intel® Compilers and Libraries for Itanium® Architecture

Heinz Bast
Heinz.Bast@Intel.com
Intel Software Enabling Group EMEA
**Intel® Software Development Products**

<table>
<thead>
<tr>
<th>Intel® Software Development Products</th>
<th>Supported Intel Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilers</td>
<td></td>
</tr>
<tr>
<td>C++</td>
<td><img src="circles.png" alt="circles" /></td>
</tr>
<tr>
<td>Fortran</td>
<td><img src="circles.png" alt="circles" /></td>
</tr>
<tr>
<td>Performance Analyzers</td>
<td></td>
</tr>
<tr>
<td>VTune™ Performance Analyzer</td>
<td><img src="circles.png" alt="circles" /></td>
</tr>
<tr>
<td>Performance Libraries</td>
<td></td>
</tr>
<tr>
<td>Math Kernel Library</td>
<td><img src="circles.png" alt="circles" /></td>
</tr>
<tr>
<td>Cluster Math Kernel Library</td>
<td><img src="circles.png" alt="circles" /></td>
</tr>
<tr>
<td>Integrated Performance Primitives</td>
<td><img src="circles.png" alt="circles" /></td>
</tr>
<tr>
<td>Threading Tools</td>
<td></td>
</tr>
<tr>
<td>Thread Checker</td>
<td><img src="circles.png" alt="circles" /></td>
</tr>
<tr>
<td>Cluster Tools</td>
<td></td>
</tr>
<tr>
<td>Trace Analyzer / Trace Collector</td>
<td><img src="circles.png" alt="circles" /></td>
</tr>
<tr>
<td>MPI Library</td>
<td><img src="circles.png" alt="circles" /></td>
</tr>
</tbody>
</table>

*Other brands and names may be claimed as the property of others.*
# Development Tools Roadmap

<table>
<thead>
<tr>
<th>Products</th>
<th>Q4 '04</th>
<th>Q1 '05</th>
<th>Q2 '05</th>
<th>2H '05</th>
<th>2006</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compilers</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C++ &amp; Ftn 8.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C++ &amp; Ftn &quot;NR&quot;</td>
</tr>
<tr>
<td>C++ for Win</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C++ Bi-Endian</td>
</tr>
<tr>
<td>CE.NET 1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C++ Tool Suite 1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>VTune™</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VTune™ 7.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VTune 8.0</td>
</tr>
<tr>
<td>VTune™ Linux *2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VTune Linux 4.0</td>
</tr>
<tr>
<td>VTune™ Update Xscale™ (Windows*)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Libraries</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MKL 7.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MKL 8.0</td>
</tr>
<tr>
<td>Cluster MKL 7.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Cluster MKL 8.0</td>
</tr>
<tr>
<td>IPP 4.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IPP 5.0</td>
</tr>
<tr>
<td>OXL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Threading Tools</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Threading Tools 2.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ITT 2.2</td>
</tr>
<tr>
<td><strong>Cluster Tools</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trace Analyzer &amp;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trace Collector 4.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel Cluster Toolkit 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Trace Analyzer 6.0 (Mosel)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Additional</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EM64T Support</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eclipse* 2.1.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Other brands and names may be claimed as the property of others.
The Intel® Compiler Family

Easiest Way to take advantage of Intel Architecture

<table>
<thead>
<tr>
<th>Language</th>
<th>Architecture</th>
<th>OS/Platform</th>
<th>Name / Name since 8.0 release</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
<td>IA32</td>
<td>Windows*</td>
<td>icl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Linux*</td>
<td>icc → ecc</td>
</tr>
<tr>
<td></td>
<td>Itanium®</td>
<td>Windows*</td>
<td>ecl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Linux*</td>
<td>ecc</td>
</tr>
<tr>
<td></td>
<td>Xscale™</td>
<td>Microsoft eMbedded Visual C++</td>
<td>ccxsccee</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Platform Builder for Win CE .NET*</td>
<td>ccxsccee</td>
</tr>
<tr>
<td>C</td>
<td>IA32 and Itanium®</td>
<td>Windows / EFI Byte Code (EBC) Virtual Machine</td>
<td>iec</td>
</tr>
<tr>
<td>Fortran</td>
<td>IA32</td>
<td>Windows*</td>
<td>ifl → ifort</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Linux*</td>
<td>Ifc → ifort</td>
</tr>
<tr>
<td></td>
<td>Itanium®</td>
<td>Windows*</td>
<td>Efl → ifort</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Linux*</td>
<td>efc→ ifort</td>
</tr>
</tbody>
</table>
EPIC Architecture Principles

1. “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

2. The Architecture should provide features that assist the *compiler* in exploiting ILP

3. The Architecture should provide mechanisms to communicate the *compiler’s* plan of execution to the hardware.”

Schlansker, Michael S. and Rau, B. Ramakrishna (HP

Intel Compiler Architecture

C++ Front End

Profiler

Interprocedural analysis and optimizations: inlining, constant prop, whole program detect, mod/ref, points-to

Loop optimizations: data deps, prefetch, vectorizer, unroll/interchange/fusion/dist, auto-parallel/OpenMP

Global scalar optimizations: partial redundancy elim, dead store elim, strength reduction, dead code elim

Code generation: predication, software pipelining, global scheduling, register allocation, code generation

FORTRAN 95 Front End

Disambiguation: types, array, pointer, structure, directives

* Other brands and names may be claimed as the property of others.
Generic Features

- Compatibility to leading open-source tools (ICC vs. GCC, IDB vs GBD)
- OpenMP 2.0 support
- Automatic parallelization
- Profile-guided optimization
- Multi-file interprocedural optimization
- Support of other Intel® tools
  - E.g –tcheck for Threading Tools

* Other brands and names may be claimed as the property of others.
Generic Features (cont.)

- Support for gprof (-p switch)
- Inlined math intrinsics
- Code Coverage Tool
- Test Prioritization Tool
Target Processor Selection

• Itanium® and Itanium®-2 processor  
  – tpp1 ( Linux ) for Itanium®  
  – tpp2 (Linux) for Itanium®-2  
  – Itanium-2 now default ( since 7.0 release )

• Current “Madison” Processor  
  – No compiler changes  
  – Same as for Itanium®-2

• Be prepared for new options for future Itanium processors ( Montecito, Tukwila)
Target Selection is relevant

```c
int csum(double a[], double b[], double c[], double x, int max)
    for (i=0; i<max;i++) a[i]=b[i]+x*c[i];
```

Intel® Itanium™ processor optimized assembler:
```assembly
.b1_2:
{   .mmi
(p16) ldfd f32=[r34],8
(p16) ldfd f37=[r33],8
    nop.l 0 ;;
} { .mfb
(p24) stfd [r32]=f46,8
(p20) fma.d f42=f8,f36,f41
    br.ctop.sptk .b1_2 ;;
}
```

Itanium™ -2 processor optimized assembler:
```assembly
.b1_2:
{   .mfi
(p16) ldfd f32=[r34],8
(p16) ldfd f37=[r33],8
(p22) fma.d f46=f8,f38,f45
    nop.i 0
} { .mmb
(p24) stfd [r32]=f46,8
(p26) stfd [r32]=f50,8
    br.ctop.sptk .b1_2 ;;
}
```

Itanium-2® processor can do 4 FP load/store operations/cycle, Itanium® only 2 !!
( double performance for large max )

L2 Latency
- Itanium: 8
- Itanium-2: 6
## Some General Switches

<table>
<thead>
<tr>
<th>Disable optimization</th>
<th>-O0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimize for speed (no code size increase), no SWP</td>
<td>-O1</td>
</tr>
<tr>
<td>Optimize for speed (default), includes SWP</td>
<td>-O2</td>
</tr>
<tr>
<td><strong>High-level optimizer</strong>, incl. prefetch, unroll, -FTZ</td>
<td>-O3</td>
</tr>
<tr>
<td>Aggressive optimizations ( == -O3 –ipo –static)</td>
<td>-fast</td>
</tr>
<tr>
<td>Create symbols for debugging</td>
<td>-g</td>
</tr>
<tr>
<td>Generate assembly files</td>
<td>-S</td>
</tr>
<tr>
<td>Assume no aliasing</td>
<td>-fno-fnalias</td>
</tr>
<tr>
<td>OpenMP 2.0 support</td>
<td>-openmp</td>
</tr>
<tr>
<td>Automatic parallelization for OpenMP threading</td>
<td>-parallel</td>
</tr>
</tbody>
</table>

* Other brands and names may be claimed as the property of others.
High-level Optimizer Overview

• Data dependence analysis
  – Array subscript analysis

• Eliminating memory operations
  – Scalar replacement, unroll-and-jam, register blocking

• Cache optimizations
  – Loop interchange, fusion, distribution, blocking, data layout

• Overlapping memory latency
  – Data prefetch

• Instruction-level parallelism
  – Unrolling, interchange, distribution

• Bandwidth optimization
  – Load-pair, array contraction

* Other brands and names may be claimed as the property of others.
Interprocedural Optimization
Extends optimizations across file boundaries

**Without IPO**

1. Compile & Optimize → file1.c
2. Compile & Optimize → file2.c
3. Compile & Optimize → file3.c
4. Compile & Optimize → file4.c

**With IPO**

1. Compile & Optimize (file1.c, file3.c, file4.c, file2.c)

<table>
<thead>
<tr>
<th>-Qip</th>
<th>Only between modules of one source file</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Qipo</td>
<td>Modules of multiple files/whole application</td>
</tr>
</tbody>
</table>
Interprocedural Optimization

• 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 predicts that $r$ and $p$ are independent making SWP possible
Profile-Guided Optimizations (PGO)

• Use 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
PGO Usage: Three Step Process

Step 1
Instrumented Compilation
```
ecl -prof_gen prog.c
```
Instrumented executable: prog.exe

Step 2
Instrumented Execution
```
prog.exe (on a typical dataset)
```
DYN file containing dynamic info: .dyn

Step 3
Feedback Compilation
```
ecl -prof_use prog.c
```
Merged DYN summary file: .dpi
Delete old dyn files unless you want their info included

* Other brands and names may be claimed as the property of others.
Programs That Benefit

• Consistent hot paths
• Many if statements or switches
• Nested if statements or switches
Profile-Guided Optimizer

Affects many compiler optimizations
- Dynamic Branch Prediction
- Loop (Pipeline vs. unroll vs. nothing)
- Cache Utilization
- Predication
- Speculation
- Classical (block order, inline, reg alloc)
- Function Splitting
Predication – The Concept

• Code Example: absolute difference of two numbers

Non-Predicated Pseudo Code

<table>
<thead>
<tr>
<th>P1:</th>
<th>cmpGE r2, r3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>jump_zero P2</td>
</tr>
<tr>
<td>sub</td>
<td>r4 = r2, r3</td>
</tr>
<tr>
<td>jump</td>
<td>end</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P2:</th>
<th>sub</th>
<th>r4 = r3, r2</th>
</tr>
</thead>
<tbody>
<tr>
<td>end:</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Predicated Assembly Code

<table>
<thead>
<tr>
<th>cmp.ge</th>
<th>p1, p2 = r2, r3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(p1)</td>
<td>sub r4 = r2, r3</td>
</tr>
<tr>
<td>(p2)</td>
<td>sub r4 = r3, r2</td>
</tr>
</tbody>
</table>

C Code

```c
if (r2 >= r3)
    r4 = r2 - r3;
else
    r4 = r3 - r2;
```

Predication Removes Branches, Enables Parallel Execution
PGO Helps Improve Predication

Do we predicate?

Balanced paths - Good opportunity to predicate independent of profile: We will double average path length but get rid of all mispredicts

* Other brands and names may be claimed as the property of others.

The Intel logo is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States or other countries.
Predication: PGO Benefits

Do we predicate?

Bad move! Main path length increased from 2 to 12 – this is 6 times and branch mispredicts are minimal.
for (i=0; i < NUM_BLOCKS; i++) {
    switch (check3(i)) {
    case 3: /* 25% */
        x[i] = 3; break;
    case 10: /* 75% */
        x[i] = i+10; break;
    default: /* 0% */
        x[i] = 99; break;
    }
}

- “Case 10” is moved to the beginning
  - PGO can eliminate most tests & jumps for the common case
Report Generation Options

- **-[Q]opt_report**
  - generate an optimization report to stderr (NB: or file)

- **-[Q]opt_report_file file**
  - specify the filename for the generated report

- **-[Q]opt_report_level level**
  - specify the level of report verbosity (min|med|max)

- **-[Q]opt_report_phase phase_name**
  - specify the phase that reports are generated against

- **-[Q]opt_report_routine name**
  - reports on routines containing the given name

- **-[Q]opt_report_help**
  - display the optimization phases available for reporting
Phase_names (from –opt_report_help)

• ipo
• ipo_inl
• ipo_cp
• ipo_modref
• ipo_lpt
• ipo_subst
• ipo_ratt
• ipo_vaddr
• ipo_pdce
• ipo_dp
• ipo_gprel
• ipo_pmerge
• ipo_dstat
• ipo_fps
• ipo_ppi
• ipo_unref
• ipo_wp
• ipo_dl
• ilo
• ilo_lowering
• ilo_strength_reduction
• ilo_reassociation
• ilo_copy_propagation
• ilo_convert_insertion
• ilo_convert_removal
• ecg
• ecg_gra
• **ecg_swp**
• ecg_predication
• ecg_speculation
• ecg_code
• ecg_code_cycles
• ecg_code_size
• ecg_code_size_fsp
• Pgo
• hlo
• hlo_fusion
• hlo_distribution
• hlo_scalar_replacement
• hlo_unroll
• hlo_prefetch
• hlo_loadpair
• hlo_linear_trans
• hlo_opt_pred
• hlo_data_trans
• hlo_reroll
• hlo_array_contraction
• hlo_scalarExpansion
• all

* Other brands and names may be claimed as the property of others.
Sample SWP Report
–Qopt_report –Qopt_report ecg_swps

Swp report for loop at line 12 in multiply_d in file multiply_d.c

- Resource II = 1
- Recurrence II = 4 >0 means loop carried dep. Rewrite loop if possible
- Minimum II = 4
- Scheduled II = 4 Min. II = Sched. II => loop optimally scheduled
- Percent of Resource II needed by arithmetic ops = 100%
- Percent of Resource II needed by memory ops = 100%
- Percent of Resource II needed by floating point ops = 100%

Number of stages in the software pipeline = 3

Following are the loop-carried memory dependency edges:
- Store at line 12 --> Load at line 12
- Store at line 12 --> Load at line 12

• What to look for:
  – Was loop software pipelined or “loop not pipelined”? 
  – Scheduled II – most important info

The Intel logo is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States or other countries.
Software Pipelining Terms

• **Initiation Interval (II)** is the number of cycles between the start of successive iterations in the loop
  • If the II is \( n \) cycles, a new loop iteration will be completed every \( n \) cycles at steady state

• **Scheduled II** is the cycles per iteration of the pipelined loop

• **Minimum II** is the smallest ii that is feasible for pipelined loop (according to the compiler’s coded uarch scheduling rules).

• **Recurrence II** is caused by loop-carried dependence edges (memory and register dependences) from instructions in one iteration to instructions in subsequent iterations
  • If the recurrences are caused by memory-dependence edges, the report prints out details of such edges
Tips on how to read SWP report

- If Recurrence II > 0, means compiler detected loop carried dependencies.
- If Minimum II = Scheduled II, means loop is optimally scheduled according to the compiler.
- Percent of Resource II used by memory ops, floating point ops and integer ops shows the utilization of the corresponding execution units throughout the loop kernel.
- If your floating point Resource II utilization is less than memory – not optimal situation for number crunching algorithm. Consider loop balancing.
Phase Ordering inside HLO

Loop Recognition and Normalization
- Construct Dependence Graph
- Loop Distribution
- Linear Loop Transformations
- Loop Fusion
- Loadpair Detection

Affine Condition Unswitching
- Block Unroll and Jam
- Loop Fusion
- Scalar Replacement
- Data Prefetching
- Loadpair Insertion

* Other brands and names may be claimed as the property of others.
Sample HLO Report

icc -O3 -opt_report -opt_report_phase hlo -opt_report_level max

... LOOP INTERCHANGE in loops at line: 7 8 9  
Loopnest permutation ( 1 2 3 ) --> ( 2 3 1 )
LOOP INTERCHANGE in loops at line: 15 17  
Loopnest permutation ( 1 2 3 ) --> ( 3 2 1 )
...
Loop at line 7 unrolled and jammed by 4  
Loop at line 8 unrolled and jammed by 4  
Loop at line 15 unrolled and jammed by 4  
Loop at line 16 unrolled and jammed by 4  
...
Loop Unrolling

- E.g. used to fully utilize execution units in each iteration
- Implicitly done by –O3
- Can be controlled by –unrollx
- Normally compiler uses a heuristic based on complexity of loop body
- Poor candidates: (similar issues for SWP or vectorizer)
  - Low trip count loops – for (j=0; j < N; j++) : N=4 at runtime
  - Fat loops – loop body already has lots of computation taking place
  - Loops containing procedure calls
  - Loops with branches
Loop Unrolling: +/-

• **Benefits**
  – Perform more computations per loop iteration
  – Reduces the effect of loop overhead
  – Can increase floating point to memory access ratio (F/M)
  – Can increase instruction level parallelism

• **Costs**
  – Register pressure
  – Code bloat (I-cache pollution)
  – Maintainability
Loop Unrolling Example

N=1025
M=5
DO I=1,N
  DO J=1,M
    A(J,I) = B(J,I) + C(J,I) * D
  ENDDO
ENDDO

Unroll outer loop by 4

K = IMOD (N,4)
DO I = 1,N-K,4
  DO J=1,M
    A(J,I) = B(J,I) + C(J,I) * D
    A(J,I+1) = B(J,I+1) + C(J,I+1) * D
    A(J,I+2) = B(J,I+2) + C(J,I+2) * D
    A(J,I+3) = B(J,I+3) + C(J,I+3) * D
  ENDDO
ENDDO
ENDDO

If loop size is known, you can eliminate post-conditioning loop by specifying the number of times to unroll

Post-conditioning loop

If loop size is known, you can eliminate post-conditioning loop by specifying the number of times to unroll
Loop Unrolling Directives

#pragma unroll
Unroll next loop; allows the compiler to determine the unroll factor

#pragma unroll(n)
Unroll next loop exactly n times

#pragma unroll(4)
for (i = 0; i < MAX; ++i) {
    a[i] = b[i] * c[i] + d[i];
}
HLO: Cache blocking

- Cache blocking ensures effective cache utilization by blocking computations within non-overlapping areas (where data references would not guarantee to reside in cache)
- Currently compiler is aware of L2 cache size, but not L3.
- Assume cache blocking for L3 needs to be done manually

* Other brands and names may be claimed as the property of others.
HLO: Loop Distribution

• Defined as: extracting different statements from a loop and distributing them into separate loops (usually with identical loop bounds)
• Used to:
  – create “tighter” code, minimize CPI
  – reduce empty cycles
  – Enable more effective usage of the CPU resources and execution units in SWP kernel
  – Reducing memory references within single loop and lower pressure on L2 OZQ
  – Reaching theoretical maximum per single loop by assisting compiler to produce balanced number of FLOPs or INT ops per load/store
  – Remove limitation on rotating register usage when SWP

* Other brands and names may be claimed as the property of others.
HLO: Loop Distribution
Triggered by Directive

#pragma distribute point

• Placed outside a loop, it tells the compiler to attempt to distribute the loop based on its internal heuristic.
• Placed within a loop, it tells the compiler to attempt to distribute the loop at that point. All loop-carried dependencies will be ignored.

```c
for (i = 0; i < MAX; ++i) {
    a1[i] = b1[i] * c1[i] + d1[i];
    #pragma distribute point
    a2[i] = b2[i] * c2[i] + d2[i];
}
```
Memory Reference Disambiguation
Options/Directives related to Aliasing

• -alias_args[-]
• -ansi_alias[-]
• -fno-alias: No aliasing in whole program
• -fno-fnalias: No aliasing within single units
• -restrict (C99): -restrict, restrict attribute
  – enables selective pointer disambiguation
• -safe_cray_ptr: No aliasing introduced by Cray-pointers
• -assume dummy_alias
• Related: Switch –ipo and directive IFDEP

* Other brands and names may be claimed as the property of others.
Vector Addition Example

```c
void add(int a[10], int b[10], int c) {
    int i;
    for (i=0; i<10; i++) {
        b[i] = a[i] + c;
    }
}
```

**C Source Code**

```assembly
.b1_7:
  {  
    .mmi
      ld4   r9=[r32],4 ;;  // cycle 0
      add   r3=r34,r9    // cycle 1
      nop.i 0 ;;        // cycle 1
  }
  {  
    .mib
      st4   [r33]=r3,4  // cycle 2
      nop.i 0          // cycle 2
      br.cloop.sptk .b1_7 ;; // cycle 2
  }
```

**ecc Compiler Output**

```
ecc -S -opt_report -opt_report_phase ecg_swp add.c
```

- loop-carried memory dependence => loop not pipelined

*Other brands and names may be claimed as the property of others.*
Enabling Compiler to SWP

void add(int *a, int * restrict b, int c, int N) {
    for(int i=0; i<10; i++)
        b[i] = a[i] + c;
}

Restrict keyword disambiguates pointer, enables compiler to SWP

ecc -S -restrict -opt_report -opt_report_phase ecg_swp add-swp.cpp

SWP report for loop at line 3 in _Z3addPiS_ii in file add-swp.cpp

| Resource II | 1 |
| Recurrence II | 0 |
| Minimum II | 1 |
| Scheduled II | 1 |

Percent of Resource II needed by arithmetic ops = 100%
Percent of Resource II needed by memory ops = 100%
Percent of Resource II needed by floating point ops = 0%

Number of stages in the software pipeline = 3
Vector Addition Example

Loop to be software pipelined:

L1: ld4 r9 = [r32],4 ;; // cycle 0
    add r3 = r34,r9 ;; // cycle 1
    st4 [r33] = r3,4 ;; // cycle 2
    br.cloop L1 ;; // cycle 2

Software pipeline stages:

stage 1: ld4 r32= [r8],4
stage 2: add r34 = r3,r33
stage 3: st4 [r2] = r35,4

Look at five pipeline iterations

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Cycle</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ld4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>Prolog</td>
</tr>
<tr>
<td>add</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X+1</td>
<td></td>
</tr>
<tr>
<td>st4</td>
<td>add</td>
<td>ld4</td>
<td></td>
<td></td>
<td></td>
<td>X+2</td>
<td></td>
</tr>
<tr>
<td>st4</td>
<td>add</td>
<td>ld4</td>
<td></td>
<td></td>
<td></td>
<td>X+3</td>
<td></td>
</tr>
<tr>
<td>st4</td>
<td>add</td>
<td>ld4</td>
<td></td>
<td></td>
<td></td>
<td>X+4</td>
<td>Kernel</td>
</tr>
<tr>
<td>st4</td>
<td>add</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X+5</td>
<td></td>
</tr>
<tr>
<td>st4</td>
<td>add</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X+6</td>
<td>Epilog</td>
</tr>
</tbody>
</table>

For 10 iterations:
SWP = 12 cycles
No SWP=30 cycles
<table>
<thead>
<tr>
<th>#pragma</th>
<th>Architecture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>swp, noswp</td>
<td>IA-64</td>
<td>Place before a loop to override the compiler’s heuristics for deciding to software pipeline the loop or not.</td>
</tr>
<tr>
<td>loop count(n)</td>
<td>IA-32, IA-64</td>
<td>Place before a loop to communicate the approximate number of iterations the loop will execute. Affects software pipelining, vectorization and other loop transformations.</td>
</tr>
<tr>
<td>distribute point</td>
<td>IA-64</td>
<td>Placed before a loop, the compiler will attempt to distribute the loop based on its internal heuristic. Placed within a loop, the compiler will attempt to distribute the loop at the point of the pragma. All loop-carried dependencies will be ignored.</td>
</tr>
<tr>
<td>unroll, unroll(n), nounroll</td>
<td>IA-64</td>
<td>Place before an inner loop (ignored on non-inmost loops). #pragma unroll without a count allows the compiler to determine the unroll factor. #pragma unroll(n) tell the compiler to unroll the loop n times. #pragma nounroll is the same as #pragma unroll(0).</td>
</tr>
<tr>
<td>prefetch a,b,…</td>
<td>IA-64</td>
<td>Place before a loop to control data prefetching. Supported when -O3 is on. #pragma prefetch a will cause the compiler to prefetch for future accesses to array a. The compiler determines the prefetch distance. #pragma noprefetch x will cause the compiler to not prefetch for accesses to array x.</td>
</tr>
<tr>
<td>ivdep</td>
<td>IA-32, IA-64</td>
<td>Place before a loop to control vectorizaton/software pipelining The compiler is instructed to ignore “assumed” (not proven) dependencies preventing vectorization/software pipelining. For Itanium: Assume no BACKWARD dependencies, FORWARD loop-carried dependencies still can exist w/o preventing SWP. Use with –ivdep_parallel option to exclude loop-carried dependencies completely (e.g. for indirect addressing)</td>
</tr>
</tbody>
</table>
C++ Compiler for Linux*

- Switch-compatible to GNU GCC for all basic options
- Object file interoperability ICC and GCC
  - “C and C” binary mix
    - No restrictions for whatever combination
  - “C++ and C” binary mix
    - No restrictions for whatever combination
  - “C++ and C++” binary mix
    - No restrictions since 8.0
    - Support for GCC STL (libstdC++ library) since 8.0
- Support of most GCC C/C++ language extensions
- Linux kernel build
  - ICC 8.1 and kernel 2.6.x: No manual changes required

* Other brands and names may be claimed as the property of others.
C++ Compiler for Linux*

Some new features

• C99 support
  – Many features but not all in 8.1

• New pragmas
  – e.g.  #pragma member alignment

• -Wcheck
  – Diagnostics for problems that could manifest as runtime problems
    • Variable “x” is used before its value is set
    • Conversion from “int” to “short” may lose significant bits
Current Release: ICC 8.1

- GCC 3.4 C/C++ Object Compatibility & Interoperability
- Linux: GCC conformance as default
  - GCC run-time libraries
- Different compiler drivers for C and C++ code
  - Like gcc and g++, there is icc and icpc
  - Will reduce code size of pure C-programs
  - Not an option – you have to use icpc for LINKING C++ application
- KMP.Schedule Environment Variable for OpenMP Scheduling Control
  - Both for C/C++ and Fortran
PCH – Pre-Compiled-Headers

– Motivation: Large Project typically contains a common subset of includes
– What: “Memory Dump” of the compiler after header file processing.
– Goal: Create the maximum subset of common header files

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Compile Time Improvement by PCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Povray* (-O2)</td>
<td>39%</td>
</tr>
<tr>
<td>Eon* (-O2)</td>
<td>29%</td>
</tr>
</tbody>
</table>
Intel® Compiler Release 9.0

• Major focus: Stability and performance improvements
• Enhanced usability for IPO – Inter-Procedural optimization
• Improved security checking
  – memory leak detection
  – bounds checking
  – Fortran routine argument checking
• Enhanced debugging of optimized code in the compiler and IDB
• Intel® C/C++ Linux IA32 Compiler Integration with Eclipse* CDT 2.0
• Support for additional GNU GCC option enhancing C++ conformance
• Intel® Itanium® Architecture stack unwinding: Mosberger stack unwinding routines will be default for compiler and debugger

Beta program started – Release planned for end of Q2/05

* Other brands and names may be claimed as the property of others.
Fortran Compiler for Linux

• Full compatibility to ANSI Fortran 77, Fortran90, Fortran95
  – Switches support language conformance

• Many extensions like
  – New options on ATTRIBUTES directives
    • DECORATE, DEFAULT
  – REAL(16), COMPLEX(16)

• Many extensions of Fortran2003 already ( former Fortran2000 )
  – Standard not approved yet
  – Some existing extensions standardized in Fortran 2003 (eg. VOLATILE)

• 8.x compiler is merger of Intel 7.x compiler and Compaq Visual Fortran 6.6
How did we get here?

Compaq Fortran 6.6

Intel Fortran 7.x

Language Features Libraries Code Gen Array Vis

Parallel

Compaq Fortran 6.6

Intel Fortran 7.x

Intel Fortran 8.x

* Other brands and names may be claimed as the property of others.
# Fortran Compiler

## Some switches to know about

<table>
<thead>
<tr>
<th>Feature</th>
<th>Switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language Standard</td>
<td><code>-f66</code> / <code>-e90</code> / <code>-e95</code> / <code>-std90</code> / <code>-std95</code></td>
</tr>
<tr>
<td>Semantic/Restrictions/Warnings</td>
<td></td>
</tr>
<tr>
<td>Data type sizes</td>
<td><code>-double_size</code></td>
</tr>
<tr>
<td>Data conversion Little-endian/Big-endian</td>
<td><code>-convert</code></td>
</tr>
<tr>
<td>(see also run-time environment variables for more selective conversion)</td>
<td></td>
</tr>
<tr>
<td>Enforce strict typing (similar to IMPLICIT NONE)</td>
<td><code>-warn declarations</code></td>
</tr>
<tr>
<td>Location of module files to be generated</td>
<td><code>-module &lt;path&gt;</code></td>
</tr>
<tr>
<td>Location of module files to be used</td>
<td><code>-l&lt;path&gt;</code></td>
</tr>
</tbody>
</table>
# Fortran Compiler
## Non-standard External Routines

<table>
<thead>
<tr>
<th>Portability routines like e.g GETENV, GETPID, SIGNAL, DTIME, ETIME … ( has been -Vaxlib before IFORT 8.0 )</th>
<th>USE IFPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSIX Routines like PXFWAIT ( wait for child process)</td>
<td>USE IFPOSIX</td>
</tr>
<tr>
<td>Only Windows compiler: Routines to access Microsoft Windows* environment</td>
<td>USE IFQWIN USE IFLOGM USE IFCOM USE IFAUTO …</td>
</tr>
<tr>
<td>Misc routines like FOR_GET_FPE and FOR_SET_FPE to get and set the floating point status registers</td>
<td>USE IFCORE</td>
</tr>
</tbody>
</table>
“Black-Belt” User’s Guide
“Black-Belt” User Guide

• Intel memo documenting numerous compiler options not listed in official product documentation useful
  – For compiler debugging
    • E.g. finding the application module where the compiler optimizer fails
  – Benchmarking/tuning in special situation

• Warning: These options are
  – Not supported
  – Not part of the usual regression testing
  – Are not intended to become “supported” options
  – Can be non-functional in future releases or have a different function in the future

• The memo is no longer Intel-confidential information but should not be made public
Black Belt: Some samples

- \texttt{mP2OPT\_hlo\_prefetch=TRUE/FALSE}
  - Enable/disable data pre-fetching
- \texttt{mP2OPT\_hlo\_distribution=TRUE/FALSE}
  - Enable/disable loop distribution
- \texttt{mP3OPT\_ecg\_mm\_fp\_ld\_latency=<value>}
  - Default=11; cache miss latency
- \texttt{mP3OPT\_ecg\_non\_ieee\_div=TRUE/FALSE}
  - Move to a faster, non-IEEE division
- \texttt{mP2OPT\_opt\_threshold\_enforce=TRUE/FALSE}
  - Consider/ignore internal limits for optimizer; (do not) give up on optimization when certain threshold for resource usage is reached

* Other brands and names may be claimed as the property of others.
Summary

• For all 4 combinations of Linux/Windows and C++/FORTRAN, Intel® offers mature compilers to develop for Itanium® architecture

• Easy way to get best performance results for Itanium®

• Wide range of options allow sophisticated and individual tuning
Intel Compiler Architecture

C++
Front End

FORTRAN 95
Front End

Profiler

Interprocedural analysis and optimizations: inlining, constant prop, whole program detect, mod/ref, points-to

Loop optimizations: data deps, prefetch, vectorizer, unroll/interchange/fusion/dist, auto-parallel/OpenMP

Global scalar optimizations: partial redundancy elim, dead store elim, strength reduction, dead code elim

Disambiguation: types, array, pointer, structure, directives

Code generation: predication, software pipelining, global scheduling, register allocation, code generation

* Other brands and names may be claimed as the property of others.
Predication: PGO Benefits

Do we predicate?

Yes:
Average path length increased from 6 to 12; mispredicts could happen each time otherwise

* Other brands and names may be claimed as the property of others.
Predication: PGO Benefits

Do we predicate?

Yes – optimal move: Average path length increased by 20% only but no mispredicts
Predication: PGO Benefits

Without profile we won’t predicate.

Not any worse than traditional architectures.

* Other brands and names may be claimed as the property of others.