Intel® Software Development Tools for the Itanium® 2 processor

Intel® software college

www.intel.com/software/college

Agenda

- Overview of Intel® Software Tools
- The sample application explained
- The Intel VTune™ Performance Analyzer
- Using the Intel C++ Compilers
- Using the Intel Libraries
- Intel Software College
- Summary

Three Main Principles of the EPIC Architecture

- "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; and
- The Architecture should provide a mechanisms to communicate the compiler's plan of execution to the hardware"

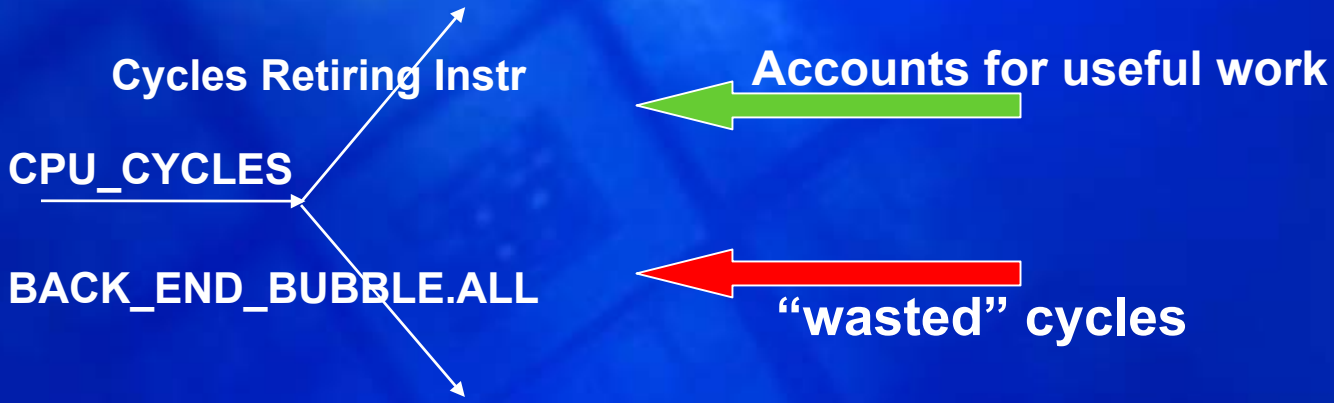
Schlansker, Michael S. and Rau, B. Ramakrishna from HP Laboratories, "EPIC: Explicitly Parallel Instruction Computing" Computer , February 2000

Performance Analysis on Itanium® 2 Processors

- The Itanium® Processor Family supports multiple new performance monitoring capabilities
  - Detailed Core Pipeline Cycle Accounting
  - Performance Event Collection Scoping
  - Event Address Register (EAR) events
- Facilitates state-of-the-art performance tuning
  - Improves developer insight into true code performance
  - Heavily utilized to quickly ramp compiler technology

Unique Features of the Itanium® Processor Family

Follow the Cycle AccountingTree



Stall Cycles Start the Analysis

Cycle Accounting Tree Determines the Architectural Events

Cycle Accounting → Architectural Events

Stall Cycles are Associated with Architectural Events

Constraining Performance Monitoring Events on IPF

- The Performance Monitoring Events can be constrained to increment by originating
  - Instruction type (opcode matching)
  - Instruction Pointer range (IP matching)
  - Virtual Address Range (Data Address matching)
  - Or any combination of the above
    - default is no constraint or collect all events

Unique Feature of the Itanium® Processor Family

Opcode Matching Examples

- Increment memory hierarchy events (cache misses etc) only for
  - Integer memory ops
  - FP loads/FP stores
  - Prefetch
- Find 1 and 2 byte memory ops
  - Locating poorly aligned memory accesses
- Find reciprocal approximations
  - Estimate FP scoreboard dependency stalls
- Tune Compiler code generation
  - Help compiler teams generate better code

EAR (Event Address Register) Events

- Identify long latency loads
- Record exact IP, Virtual Address and data delivery latency in Hardware
  - Sampled subset of all loads
- Unambiguously identifying slow data accesses in code
- Powerful tool for developing prefetch strategies in compilers

Unique Feature of the Itanium® Processor Family

Overview

Intel® Software Development Products

- **Intel Compilers**
  - Intel C++ 7.1 for Windows* and Linux*
  - Intel Fortran 7.1 for Windows/Linux
  - All support Pentium® 4 Processor, Itanium® Processor, Itanium® 2 Processor, Intel Xeon™ processor
  - Supports latest Microsoft .Net* development environment
  - Compatible with gcc
- **Intel Libraries**
  - Intel Integrated Performance Primitives (Intel IPP), Intel Math Kernel (Intel MKL) support Windows and Linux
  - Intel IPP SA, supports WinCE*
  - Intel IPP for Intel XScale™ microarchitecture
- **Intel VTune™ Performance Analyzer**
  - 7.0, Pentium 4 Processor support, Itanium 2 Processor, Windows and Linux support
- **Intel Threading Tools**
  - Intel® Thread Checker
  - VTune™ Analyzer Thread Profiler
  - Parallel Applications Center, threading lab in Champaign, Illinois
- **Intel VTune™ Enterprise Analyzer**
  - Performance analysis tool for the MS n-tier infrastructure
  - Gathers information on network traffic
  - Measures system resources

Agenda

- Overview of Intel® Software Tools
- The Sample Application Explained
- The Intel VTune™ performance analyzer
- Using the Intel C++ Compilers
- Using the Intel threading tools
- Using the Intel Libraries
- Intel Software College
- Summary

The Sample Application

- Modified “C” version of Linpack benchmark
  - Solves a linear system of equations
    - `Ax=B`
      - First does LU decomposition on A
        - `dgefa`
      - Then solves for x
        - `dgesl`
  - Reports results in millions of floating point operations per second (MFLOPS)

Sample Application

Disclaimer:

- Results reported are not meant to be used as an official benchmark report
- Meant to show tool usage not System performance
- The System: 4 Intel® Itanium® 2 Processors at 1 Ghz

The Sample Application

Baseline Performance: How Does GCC Do?

- Various options used:

  -s -static -O3 -fomit-frame-pointer -funroll-loops -fexpensive-optimizations -fschedule-insns2 –ffast-math

- 76 MFlops GCC 2.96 : -O2
- 77 MFlops GCC 3.2.2 : Max Options

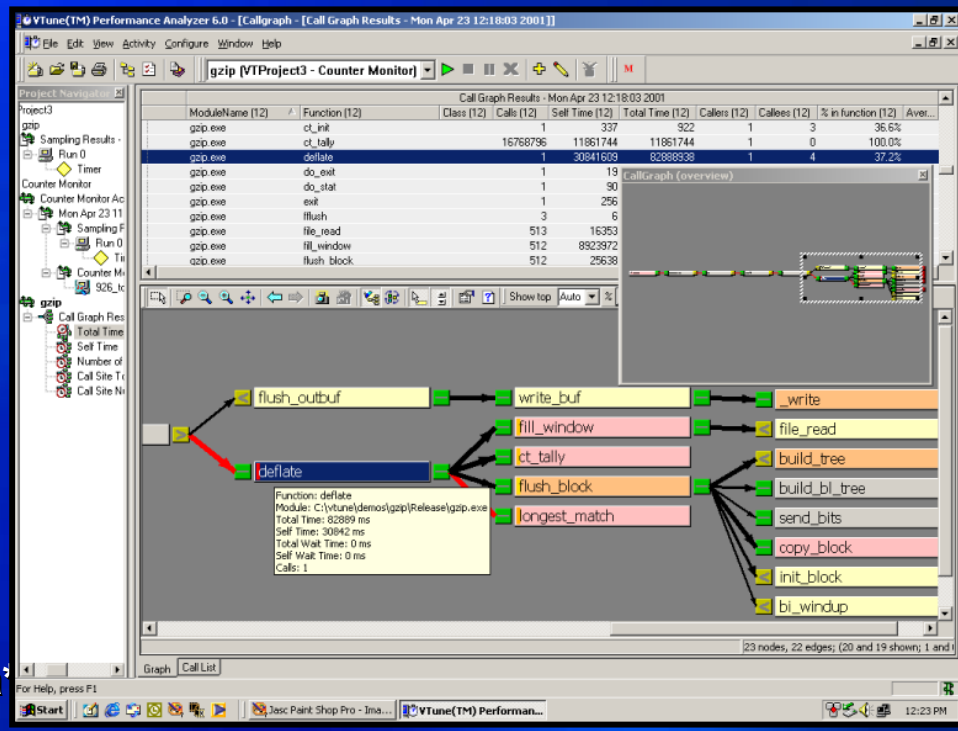
Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, visit http://www.intel.com/performance/resources/limits.htm.

Agenda

- Overview of Intel® Software Tools
- The Sample Application Explained
- The Intel VTune™ Performance Analyzer
- Using the Intel C++ Compilers
- Using the Intel threading tools
- Using the Intel Libraries
- Intel Software College
- Summary

Intel® VTune™ Performance Analyzer 7.0

- **Available now**
- **Features:**
  - Supports Pentium® 4, Intel® Xeon™ and Itanium® 2 processors, and the Mobile Intel Pentium III Processor-M
  - Linux* remote data collectors (32- and 64-bit)
  - Multi-threading support and Hyper-threading on processors
  - Adds .Net* and C# support
    - **Also supports C/C++, Fortran, Java* SDK 1.3 & 1.4, Assembler, Visual Basic***



- **Identifies performance bottlenecks in source code with <u>low intrusion</u> event and time based technology**
  - **Interrupt-based sampling using CPU registers**

VTune™ Performance Analyzer in The Linux* Environment

- Collection
  - Remote collector on Linux
    - Remote sampling driver compiled into kernel
    - Then start VTServer
- Analysis
  - VTune Performance Analyzer GUI runs in Windows*

VTune™ Performance Analyzer

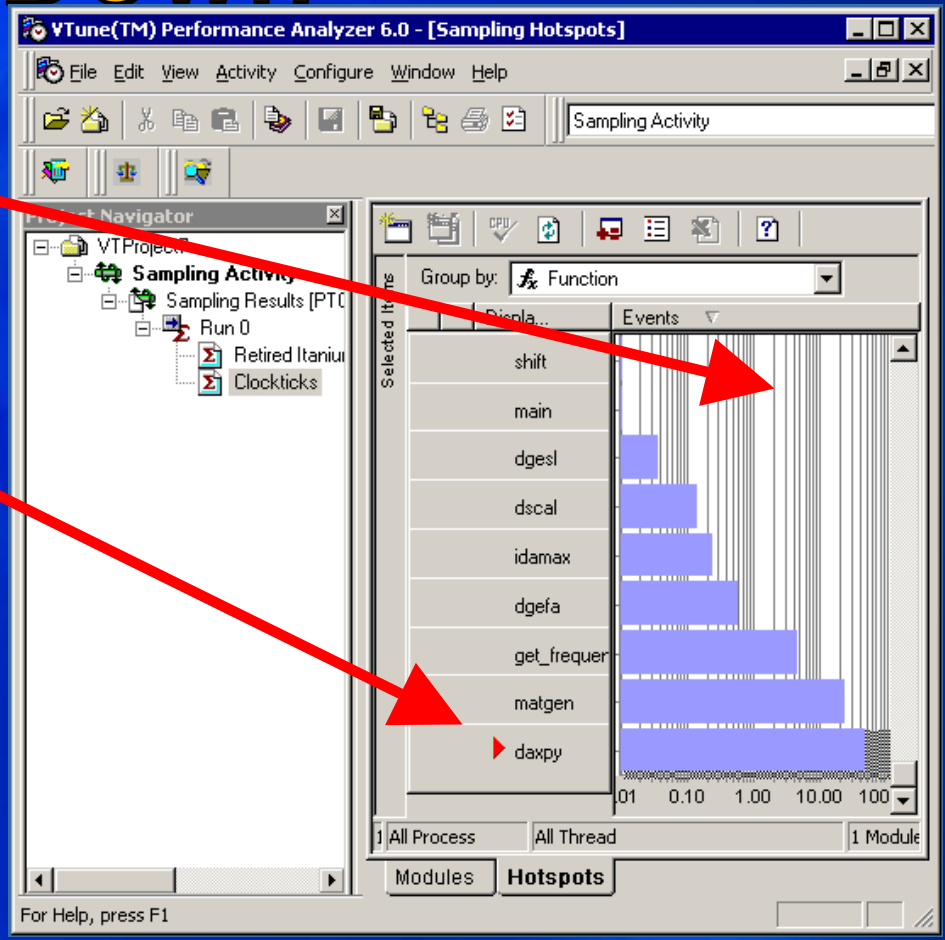
Hotspot Drill Down

Log scale

Most samples within daxpy

Use in combo with gprof for better understanding

Call Graph for Linux* - Coming Soon!



Agenda

- Overview of Intel® Software Tools
- The sample application explained
- The Intel VTune™ Performance Analyzer
- Using the Intel C++ Compilers
  - Switches for Itanium™ processor
  - High level optimizer
  - Inter-procedural optimizations
  - Profile guided optimizations
  - Report creation
  - Sample application performance
  - Auto-parallelization
  - OpenMP
- Using the Intel Libraries
- Intel Software College
- Summary

Intel® Compilers 7.0

- **Newly Released version!**
  - C++ and Fortran
  - IA-32 and Intel® Itanium™ processor-based systems
  - Windows* or Linux*
- **Advanced optimization features**
  - IA32 specific features such as support for SSE2, Intel® NetBurst™ microarchitecture, and Automatic Vectorization
  - Itanium specific features like Branch Prediction and Software Pipelining
- **Threaded application support (Hyper-Threading Technology)**
  - OpenMP* 2.0 standard support
  - Auto-Parallel feature that automatically generates threaded code for parallel loops
- **Linux Specific:**
  - Source/Binary compatibility with GNU C
  - C++ ABI conformance
  - Ability to build the Linux Kernel with minor modifications
- **Windows specific:**
  - Integrates into MS Visual Studio (6.0 or .NET*) IDE
  - Support for MSVC.NET* language features (no support for C# or managed code)
  - Substantial native source and object code compatibility with MSVC++* 6.0 & .NET

Advance Compiler Optimization

- HLO – High Level Optimizer (-O3)
  - Enables Loop Unrolling & Loop Interchange
  - Aggressive SW Pipelining
- IPO – Interprocedural Optimizations
  - Inlining across code modules
  - "Whole Program" optimization
    - Pointer Alias Disambiguation

Advance Compiler Optimization

- PGO – Profile Guided Optimization
  - Use execution-time feedback to guide opt
  - Helps I-cache, paging, branch-prediction
    - Basic block ordering
    - Better register allocation & function inlining
    - Branching vs. Predication

Compilers

Speedup in Linpack

- Max GCC
  - 77 Mflops
- Intel Compiler at -O2
  - 146 Mflops
- Intel® compiler switches
  - 414 Mflops

OpenMP*

- OpenMP: An API which supports multi-platform shared-memory parallel programming in C/C++ and Fortran on most architectures (OS and CPU)

- Very simple example:

```
#pragma omp parallel for
 for (i = 0;i < n; i++) {
       dy[i] = dy[i] + da*dx[i];
 }
```

- Switch: -openmp

Parallelization of Linpack

- Could parallelize daxpy
  - Careful analysis indicates that DAXPY is memory bound not CPU bound
  - No speedup found
- Parallelize at highest level possible
  - Lets you choose the functions that call DAXPY (dgefa and dgesl)
  - Primary loop in dgefa is loop dependant
  - Parallelize the loop which called daxpy

Speedup in Linpack

- Intel® compiler switches
  - 414 Mflops
- OpenMP + Switches:
  - 2,923 Mflops

- Better approach: Find parallel LU decomposition algorithm
- Even Better approach: Get the Intel Parallel Applications Center to parallelize my application
- Best approach: Use Intel Math Kernel Library!

Agenda

- Overview of Intel® Software Tools
- The sample application explained
- The Intel VTune™ Performance Analyzer
- Using the Intel C++ Compilers
- **Using the Intel Libraries**
- **Intel Software College**
- **Summary**

Intel Math Kernel Library

- Four Basic APIs
  - Basic linear algebra system (BLAS) 1, 2, and 3
    - Basic matrix/vector/scalar operations
  - LAPACK (linear algebra package)
    - Solvers and eigenvector/eigenvalue solvers
  - VML (vector math library)
    - Transcendentals (sin,abs) on large data sets
  - FFT (fast Fourier transform)

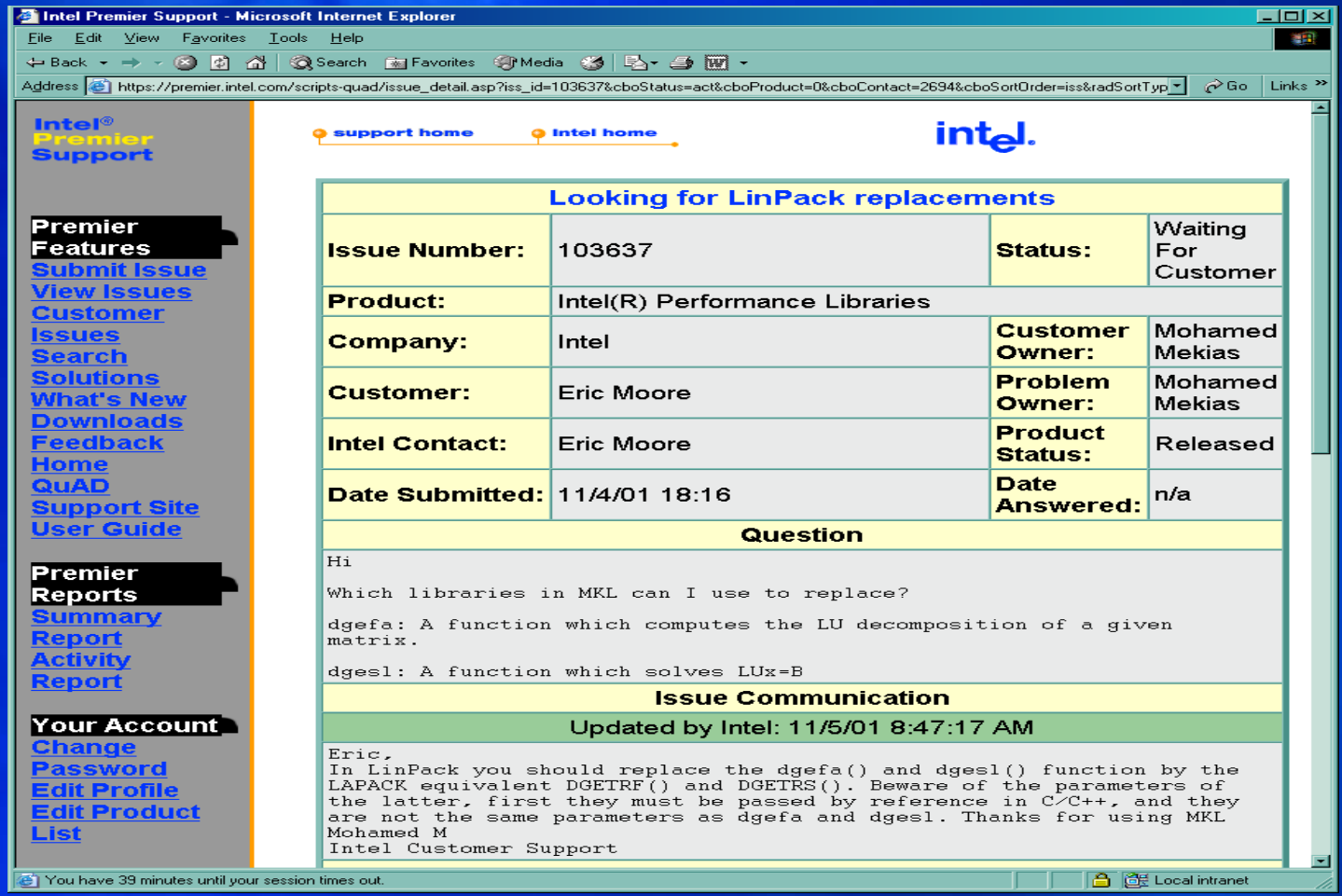
Adding Intel MKL to LinPack

- Could directly replace with BLAS:
  - daxpy, idamax, dscal, and ddot.
- No direct replacement:
  - dgefa or dgesl – the main routines!

Let's contact Intel support!

Intel® Performance Libraries

Intel MKL Support



Replace and Compile

- Replace:

```
            t1 = second();
#ifdef MKL
            DGETRF(&lda,&lda,a,&lda,ipvt,&info);
#else
            dgefa(a,lda,n,ipvt,&info);
#endif
            atime[0][0] = second() - t1;
            t1 = second();
#ifdef MKL

   DGETRS(&trans,&n,&nhrs,a,&lda,ipvt,b,&lda,&info
   );
#else
            dgesl(a,lda,n,ipvt,b,0);
#endif
            atime[1][0] = second() - t1;
```

Intel® Performance Libraries

New C Linpack Linux* Performance

9,701 Mflops

125x performance increase

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, visit http://www.intel.com/performance/resources/limits.htm.

Agenda

- Overview of Intel® Software Tools
- The sample application explained
- The Intel VTune™ Performance Analyzer
- Using the Intel C++ Compilers
- Using the Intel® threading tools
- Using the Intel Libraries
- **Intel Software College**
- **Summary**

Intel® Software College

- **CISCO, HEWLETT PACKARD, COMPAQ**

  "Each of the three courses gave valuable insights on the latest Intel technologies and products. My time and money could not have been used better. We are expecting a two-fold performance boost in some of our software products using Intel's tools."

  Amit Pabalkar, Software Engineer, California Digital

- **Expert Instructors**
  - Practicing experts in each field
  - Certified on each course
- **Flexible Delivery**
  - Worldwide Training Centers
  - Onsite delivery and customized courses available
  - ***Online courses available anytime anywhere***
- **Courses cover:**
  - Tools: Compilers, VTune™ analyzer 6.0, Libraries, Threading Tools, EFI
  - Platforms: Pentium™ 4, Intel® Xeon™, Itanium® (and Itanium 2) processors
  - Operating Systems: Windows*, Linux*

What's New in the College?

Thread Programming and Hyper-Threading Technology

New

- Develop and improve thread programming skills
- Learn about different threading models and methodologies
- Gain hands-on experience on the most efficient techniques for developing well optimized threaded applications for Hyper-Threading and multi-processors

High Performance Computing: Clustering Workshop

New

- Build hands-on experience in cluster system setup, parallel programming models, and tuning for single-node performance
- Program and optimize for clusters using shared-memory nodes with OpenMP* and distributed memory / cluster systems with MPI

Tuning for the Intel® Itanium® 2 Architecture

Updated

- Gain a thorough understanding of the Itanium® 2 processor and platform architecture
- Build expertise in 64 bit application development and optimization through lecture and hands-on labs

For the complete course catalog visit **http://www.intel.com/software/college**

Agenda

- Overview of Intel® Software Tools
- The sample application explained
- The Intel VTune™ Performance Analyzer
- Using the Intel C++ Compilers
- Using the Intel® threading tools
- Using the Intel Libraries
- Intel Software College
- **Summary**

Summary

- Use Intel® Software Development Tools to help maximize performance with less effort
- Consider use of the following tools for Windows & Linux
  - Intel VTune™ Performance Analyzer
  - Intel C++ & Fortran compilers
  - Intel Threading Tools
  - Intel Performance Libraries
- Intel Software College provides detailed training

Reference

For Further Information: Evals, Training and Support

- Web-based and classroom training
  www.intel.com/software/college
- Evaluation Versions of Software Tools
  www.intel.com/software/products
- Product & Developer support resources
  www.intel.com/software/products/support
  www.intel.com/ids

Backup

Intel® software college

www.intel.com/software/college

References

Isom L. Crawford, Kevin R. Wadleigh, “Software Optimization for High Performance Computers”, Prentice Hall PTR, 05/2000

Krishnaiyer, Rakesh…, “An Advanced Optimizer for the IA-64 Architecture”, IEEE Micro, December 2000

Schlansker, Michael S. and Rau, B. Ramakrishna, “EPIC: Explicitly Parallel Instruction Computing” IEEE Micro , February 2000

Moore, Eric W, “Intel Software Development Tools for Linux” April 2002 - Intel Senior Software Engineer

Intel® Compiler Products

- Intel® C++ for Windows*
  - Plugs into Microsoft* Visual Studio* 6.0 & .NET
  - Object compatible with Visual C++*
- Intel® C++ for Linux*
  - Uses Linux* tool-chain for development
  - Object compatible with gnu C and supports the C++ ABI
  - Support for GNU in assembler on IA-32 processors
- Intel® Fortran for Windows*
  - Plugs into Microsoft* Visual Studio* 6.0 & .NET*
  - Substantially compatible with Compaq* Visual Fortran (CVF) with important 'bridge' features for CVF users
- Intel® Fortran for Linux*
  - Uses Linux* tool-chain for development, including Intel-sponsored gdb Fortran 95 (http://gdbf95.sourceforge.net/)
  - Substantial compatibility with CVF to support portability

Microsoft Windows xp

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

intel

Intel software college

Intel® Compilers 7.1 Features

- Support latest Intel processors: Intel® Pentium® 4, Xeon™ and Itanium® 2 processors, including Intel® Pentium® M
  - Pentium processor support: SSE2, Net-Burst Architecture, Hyper-threading, Intel® Centrino™ mobile technology
  - Itanium® 2 processor support: Takes advantage of software pipelining, improved branch prediction, branch reduction thru predication
- Advanced optimization features of Intel compilers
  - Profile Guided Optimization (PGO), Inter-Procedural Optimization (IPO)
  - Parallelism: Auto-parallelization, vectorization, support for OpenMP*
  - Data prefetching
  - Processor dispatch on IA-32 processors
- Intel Premier Support: Secure internet connection directly into Intel
  - Support
  - Updates
  - Information

Intel® Compilers 7.1 Features

- Supported Linux* Platforms: 2.4 kernels
  - Pentium processor support: glibc 2.2.5, 2.2.93 (includes Red Hat* 8.0, SuSE 8.0)
  - Itanium processor support: glibc 2.2.4, 2.2.5 (includes Red Hat* Advanced Server 2.1, United* Linux* 1.0 / SuSE* Linux* Enterprise Server 8.0)
- Supported Windows* Platforms
  - Pentium processor support: Win2K-32, Win98/SE, WinNT 4.0, Windows* ME, Windows* XP, Windows.Net*Server
  - Itanium® processor support
    - Cross compilers: Win2K-32, Win98/SE, WinNT 4.0, Windows* ME, Windows* XP, Windows.Net*Server, Platform SDK
    - Native compilers: Win2K-64, Windows.Net Server

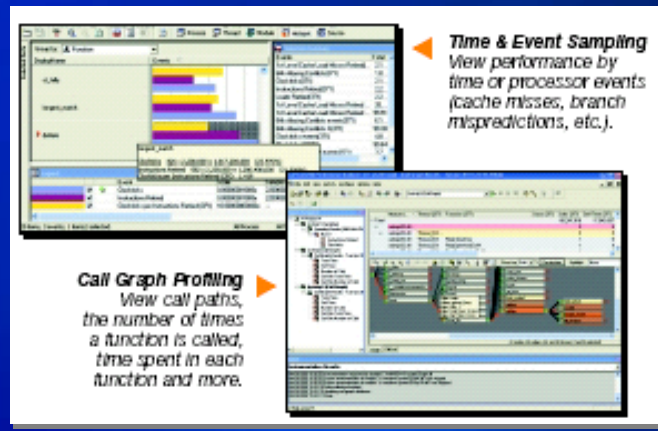
intel®

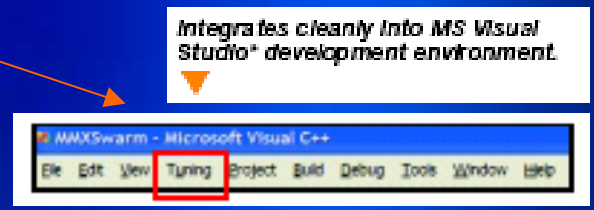
*Other names and brands may be claimed as the property of others.

Intel® software college

VTune™ Performance Analyzer 7.0

- Allows you to save time in the development cycle by quickly identifying “hot spots” for review
- Identifies performance bottlenecks in Source Code using three modes
  - 1.) Sampling – events and time based
  - 2.) Call Graph – presents program flow
  - 3.) Counter Monitor – monitors process against the CPU
- Supports latest Intel® processors incl. Itanium® 2, Pentium® 4 and Centrino™ mobile technology
- New Features:
  - Visual Studio* integration
  - Linux* remote call graph data collector (32-bit only)
  - Windows* command line interface for sampling
  - VTune Analyzer Driver Kit for unsupported Linux distributions & kernels
  - Intel® XScale™ technology support (PXA250 processor using Win CE*)
  - HT enhancements with comparison view





Process

- Step 1)
  - Binary placement/access
    - Accessibility from both Windows* and Linux*
- Step 2)
  - Start VTServer
- Step 3)
  - Launch Windows* GUI
- Step 4)
  - Instruct GUI to start sampling application

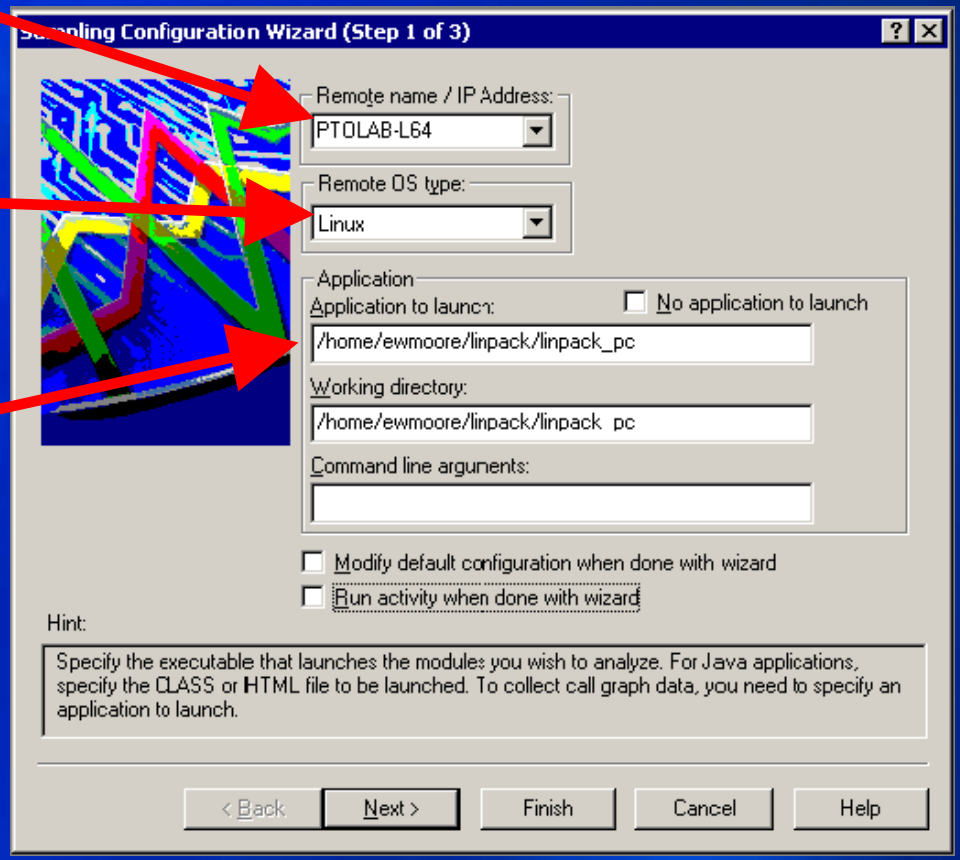
VTune™ Performance Analyzer

Sampling Wizard Step 1

Remote Host Name

Choose Linux*,
Choose Itanium® architecture

/Path/Binary



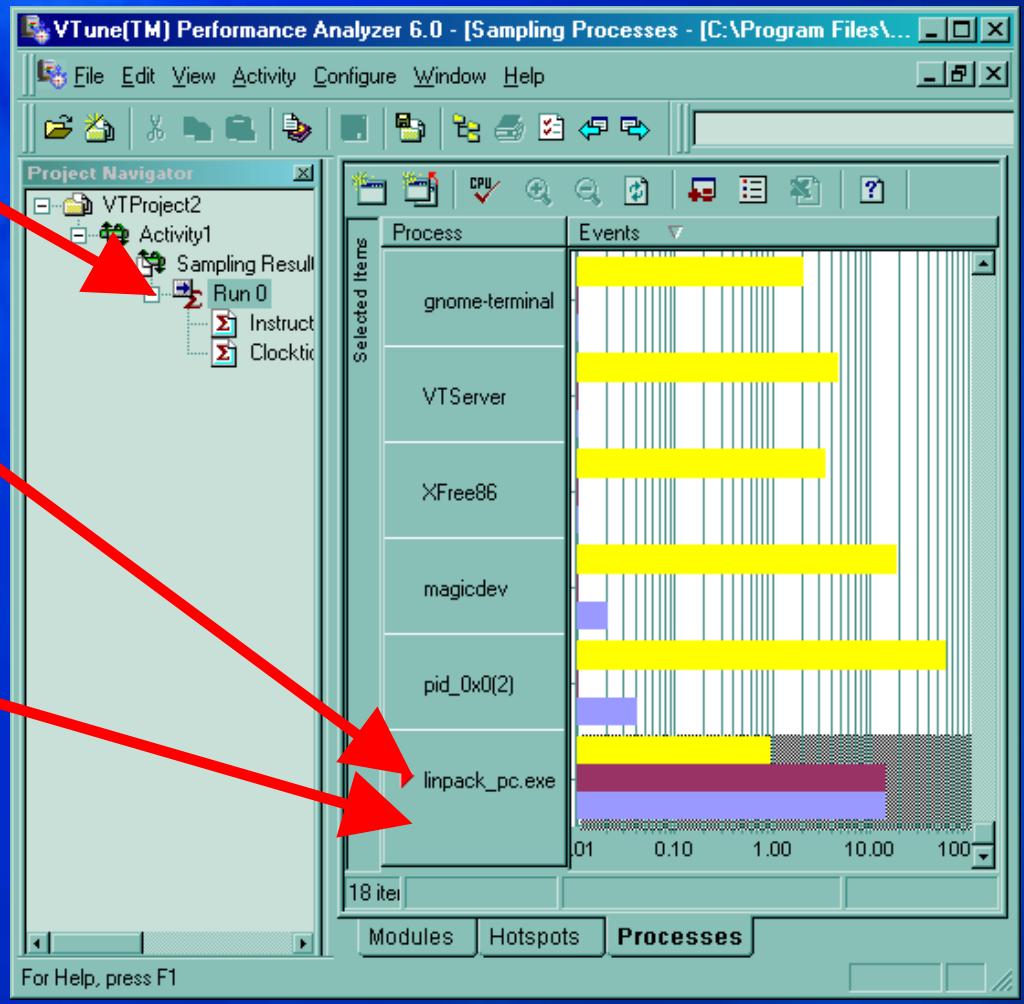
VTune™ Performance Analyzer

Analyze the Results

Choose Project/Activity/Run

View Hotspot

Most Clockticks

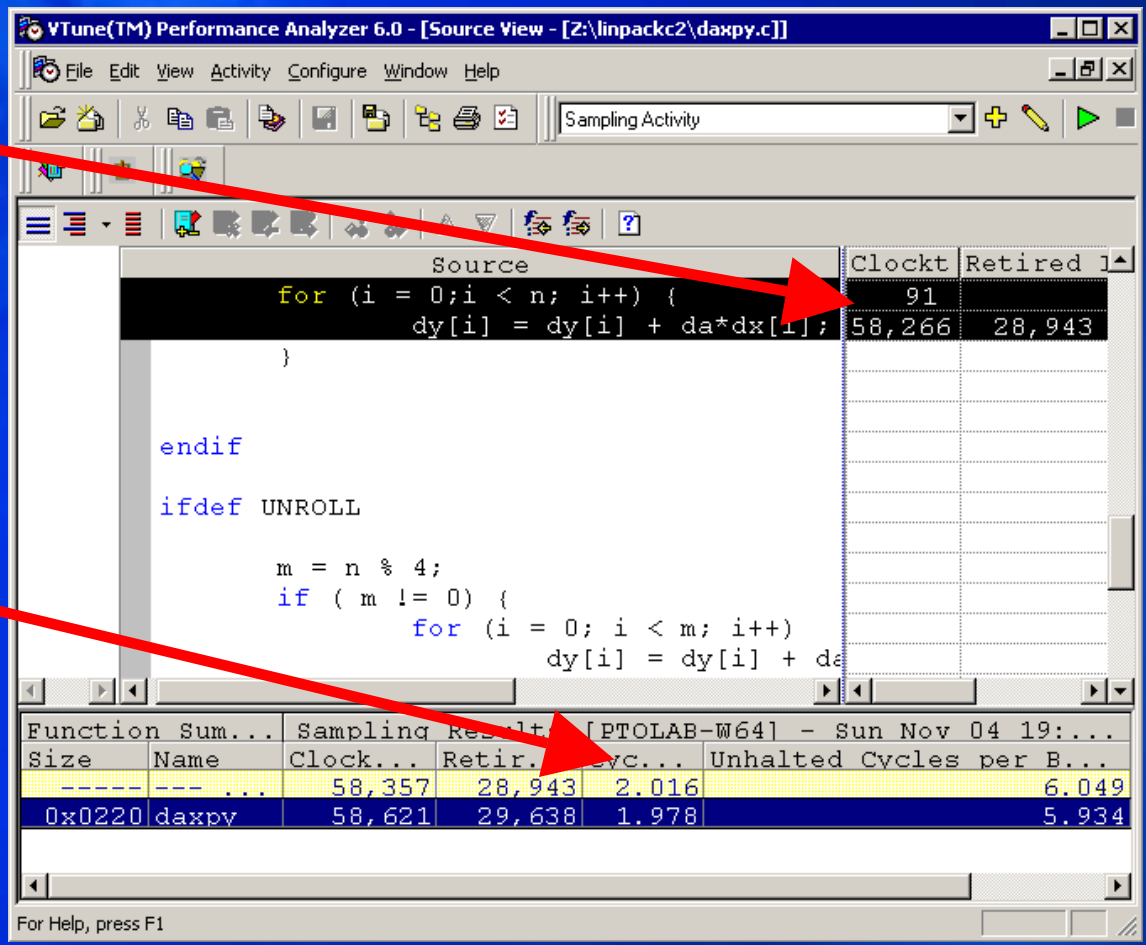


VTune™ Performance Analyzer

Source Level View

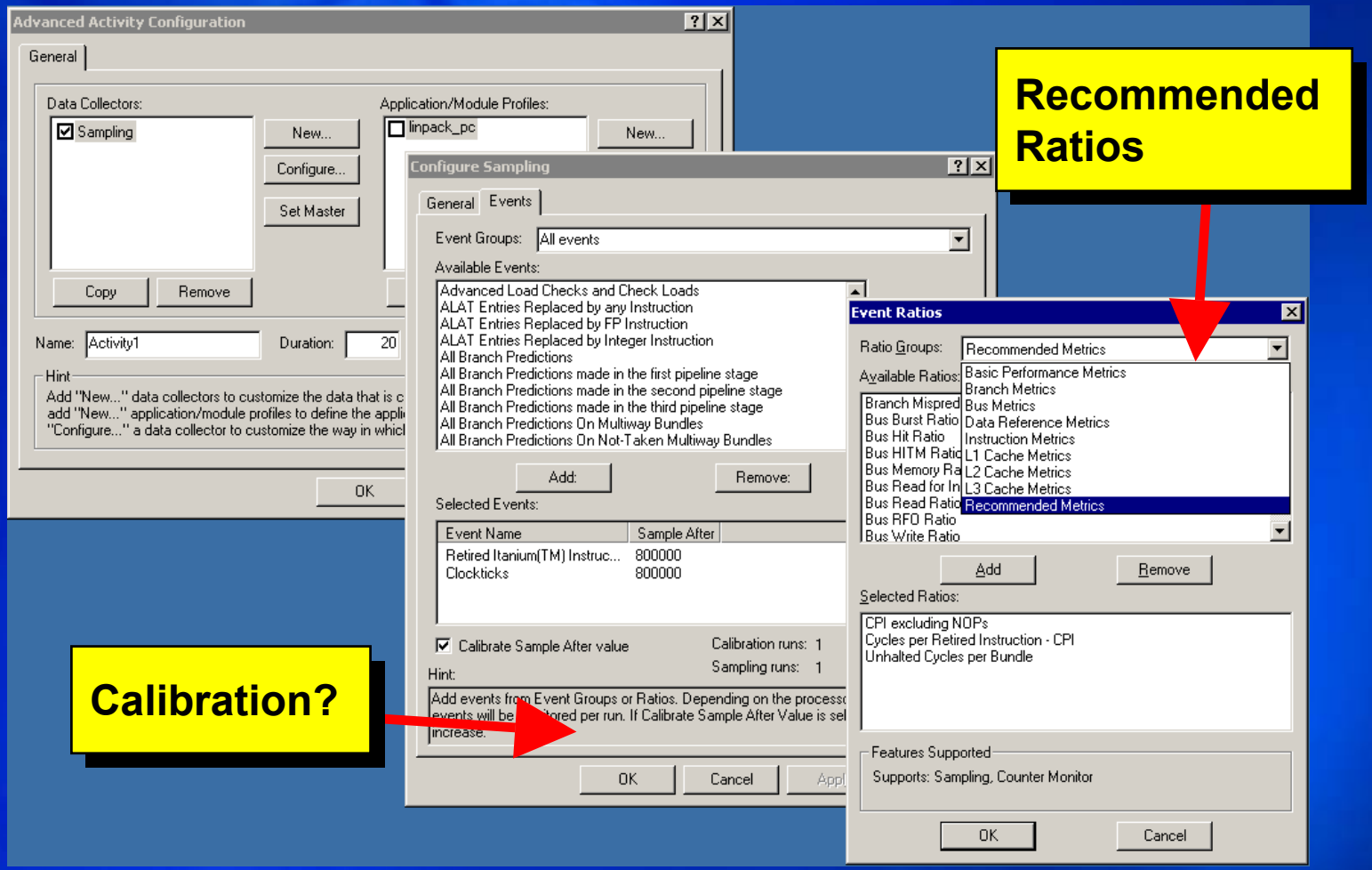
HotSpot

Efficiency?



VTune™ Performance Analyzer

Advanced Sampling



High-Level Optimizer

- Overview
  - Loop level optimizations
  - Converts source code algorithm to use more optimal memory access pattern
- How
  - Linux*: -O3
- Loops must meet certain criteria…
  - Iteration independence
  - Memory disambiguation
  - High loop count

High Level Optimizer

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

turns into:

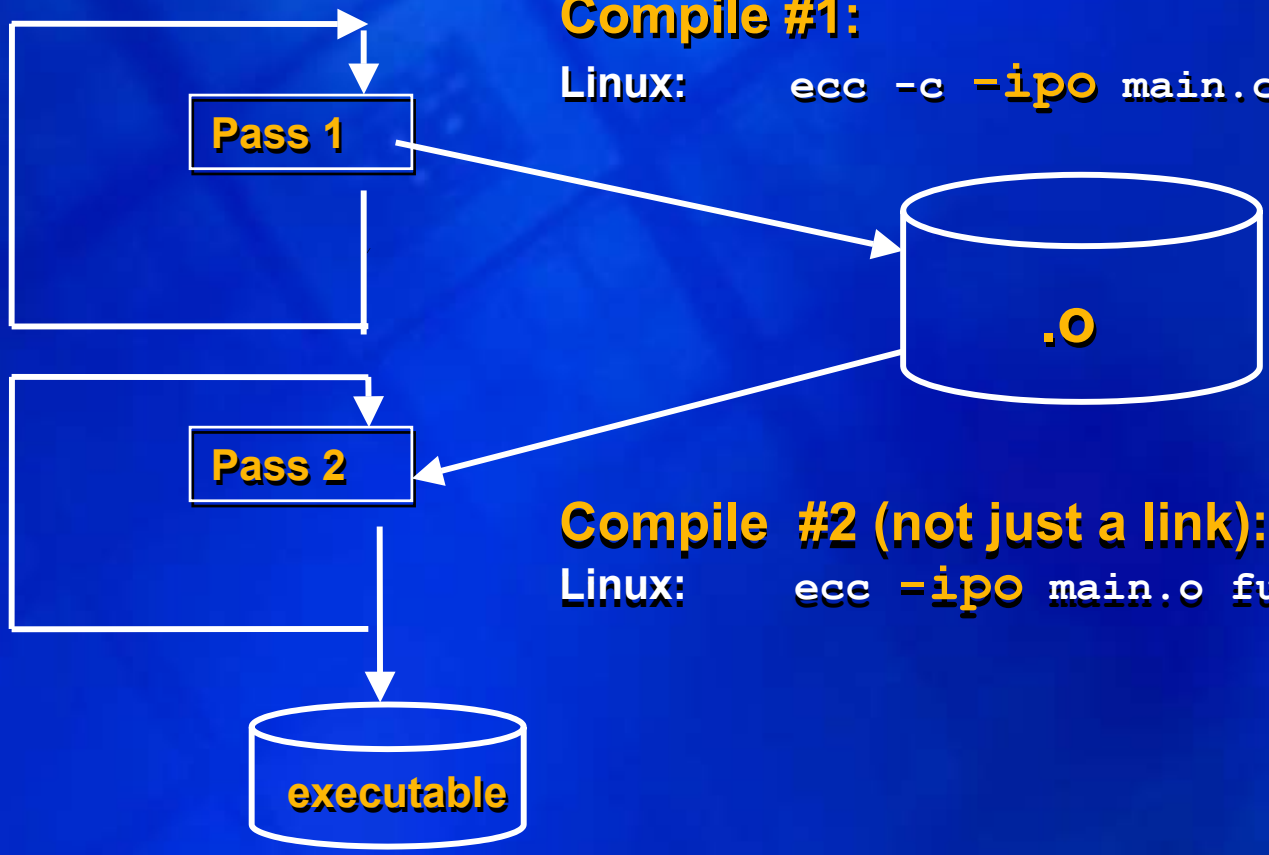
```
for (j=1; j<1000; j+=2) {
  y(j) = y(j) + a*x(j)
  y(j+1) = y(j+1) + a*x(j+1)
}
```

- Now it can use ldfp instead of ldf
- Unroll again based on number of memory ports

Inter-Procedural Optimizations (IPO)

- -ip: Enables interprocedural optimizations for single file compilation
- -ipo: Enables interprocedural optimizations across files

IPO Usage: 2 Step Process



Compile #1:

Linux: `ecc -c -ipo main.c func1.c func2.c`

Compile #2 (not just a link):

Linux: `ecc -ipo main.o func1.o func2.o`

Profile-Guided Optimizations (PGO)

- Use execution-time feedback to guide opt
- 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
Windows*: icl /Qprof_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
Windows: icl /Qprof_use prog.c

→ **Merged DYN Summary File: .dpi**
Delete old dyn files unless you want their info included too

Creating the Reports

- Which loops optimized?
- Which criteria did my loop not meet…
- OpenMP*, SWP and HLO
- How

  Linux*:

```
-opt_report
-openmp_report
```

```
-
opt_report_help
```

Software Pipelining Report

```
Daxpy.c: 42-44
 for (i = 0;i < n; i++) {
                dy[i] = dy[i] + da*dx[i];
        }

--------------------------------------------------------------------------
Swp report for loop at line 43 in daxpy in file daxpy.c

        According to the estimate of the Modulo Scheduler, the acyclic
        global scheduler can achieve a better schedule than software
        pipelining. Perhaps this loop has too many IF statements, or
        it has a loop-carried memory dependence => loop not pipelined

        Following are the loop-carried memory dependence edges:
        Store  at line    43 --> Load   at line    43
--------------------------------------------------------------------------
```

Pointer Disambiguation

- Pointer disambiguation
- -fno-alias
  - Assume no aliasing on all pointers
- -restrict
  - Use the “restrict” keyword on any pointer that is not aliased

Compilers

Auto-Parallelization

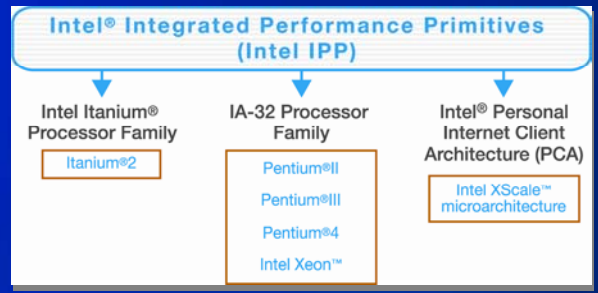
- Automatically converts loops to use multiple processors
- Enabled through
  - Linux*: -parallel
- Loops must meet certain criteria…
  - Iteration independence
  - Memory disambiguation
  - High loop count

dgefa with OpenMP* Directives

```
#pragma omp parallel omp for private(t)
for (j = kp1; j < n; j++) {
   t = a[lda*j+l];
   if (l != k) {
      a[lda*j+l] = a[lda*j+k];
      a[lda*j+k] = t;
   }
   daxpy(n-(k+1),t,&a[lda*k+k+1],1,
      &a[lda*j+k+1],1);
}
```

Intel Integrated Performance Primitives (Intel IPP)

- A library of signal, image, graphic, multimedia and numeric processing functions
- One API across Intel architectures
- Highly-optimized, processor-specific code
  - Optimized for application performance on Intel Pentium® 4, Intel® Pentium M processor component of Intel® Centrino™ mobile technology, Xeon™ and Intel® Itanium ™ 2 processors, plus Intel® Personal Internet Client Architecture (Intel PCA) applications processors based on Intel® XScale™ technology.

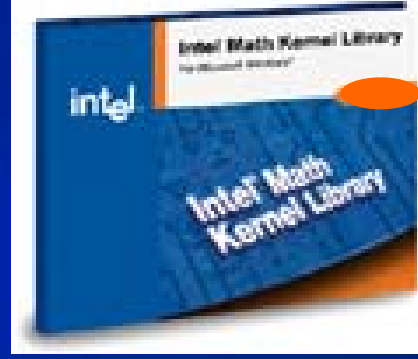


Benefits

- Pre-built library functions enable developers to focus on building value-add functionality, speeding application time to market, saves development costs
- Apps use single API - no hand-coding for processor functions
- Provides optimized performance across Intel® processors

Intel Performance Libraries:
Write once, realize the performance of your SW over many processor generations

Intel® Math Kernel Library Version 6.0



- A library of numerical processing functions highly optimized for math, scientific, engineering and financial applications
- Pre-built library functions enable developers to focus on building value-add functionality, speeding application time to market, saves development costs
  - Linear Algebra: LAPACK plus BLAS (Levels 1, 2, 3)
  - Discrete Fourier Transforms (DFT)
  - Vector Statistical Library functions (VSL)
  - Vector transcendental math functions (VML)
- New Functionality:
  - New Discrete Fourier Transforms (DFTs) commonly used in digital signal processing and image processing simulation and modeling
    - Extended multidimensional transforms beyond 1D & 2D → up to 7D
    - Mixed radix support (not just radix-2)
    - Multiple 1D FFTs on single call
  - New Vector Statistical Library (VSL), random number generators for accelerating Monte Carlo simulations such as used in physics, chemistry, medical simulations as well as financial analysis software

Intel® Math Kernel Library Version 6.0 (Continued)

- Provides optimized performance across Intel® processors
  - Intel® Pentium® 4, Intel® Pentium M processor component of Intel® Centrino™ mobile technology, Intel® Xeon™ and Itanium®2 processors
  - Version 6.0 improvement cases include:
    - BLAS dgemm small matrix 10X improvement
    - BLAS dgemm large matrix ~15% improvement
  - New dispatching static libraries offer run-time selection of processor for best performance
  - Multi-threading support using OpenMP*, Thread safe library
- Available for Microsoft* Windows & Linux operating systems

Intel Performance Libraries:
Write once, realize the performance of your SW over many processor generations

InterProcedural Optimizer

One example of 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;
    c = L * *r++ + (c>>16); *p++ = c;
    c = L * *r++ + (c>>16); *p++ = c;
    c = L * *r++ + (c>>16); *p++ = c;
}
```

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

Are *r and *p independent?

Should we disambiguate?

InterProcedural Optimizer

Memory disambiguation / data speculation

- r and p are probably independent

```
ld r1 = *r
mul r3 = L * r1
```

Advanced load

```
ld.a r21 = *r
mul r23 = L * r21
```

Loads/checks have cost.

```
shift r4 = (c >>16)
add c = r3 + r4
st *p = c
```

Check for earlier writes to same location.

```
shift r24 = (c>>16)
chk.a r21,L3
add c = r23 + r24
st *p = c
```

```
ld.a r31 = *r
mul r33 = L * r31
```

```
shift r34 = (c>>16)
chk.a r31,L4
add c = r33 + r34
st *p = c
```

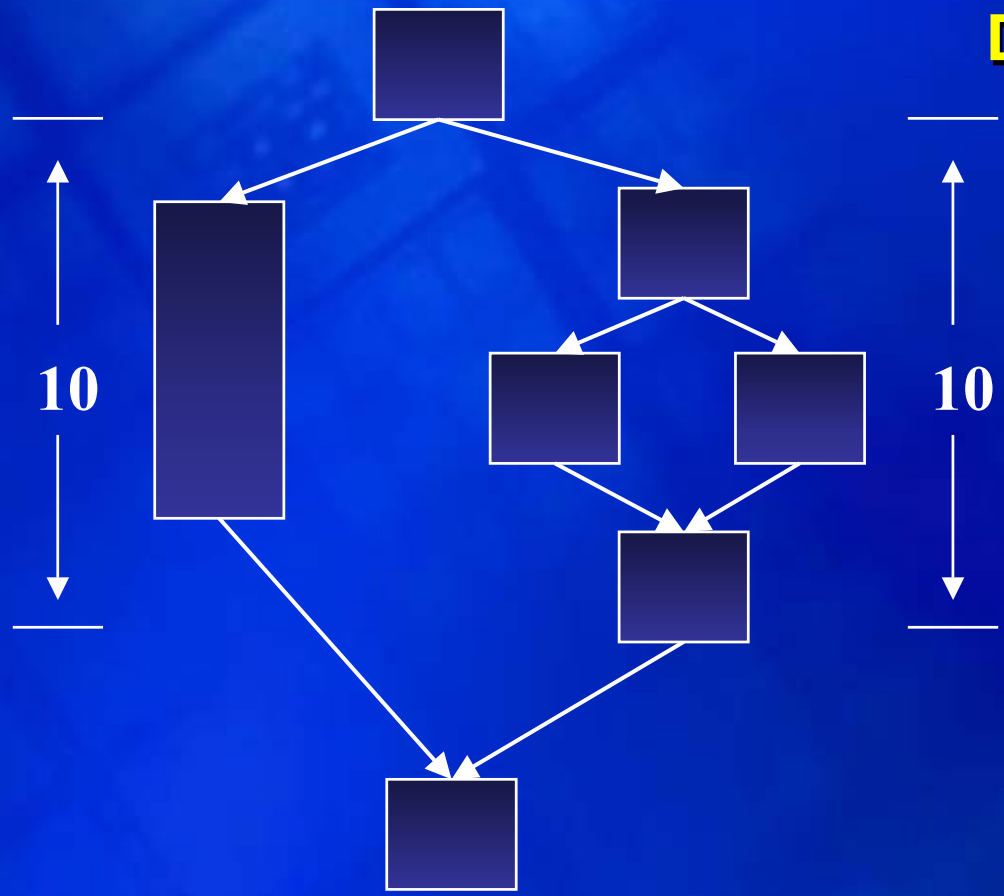
Profile Guided Optimizer

Affects many compiler optimizations

- Predication
- Speculation
- Cache utilization
- Loop (pipeline versus unroll versus none)
- Classical (block order, inline, reg alloc)
- Function splitting

Compilers

Predication

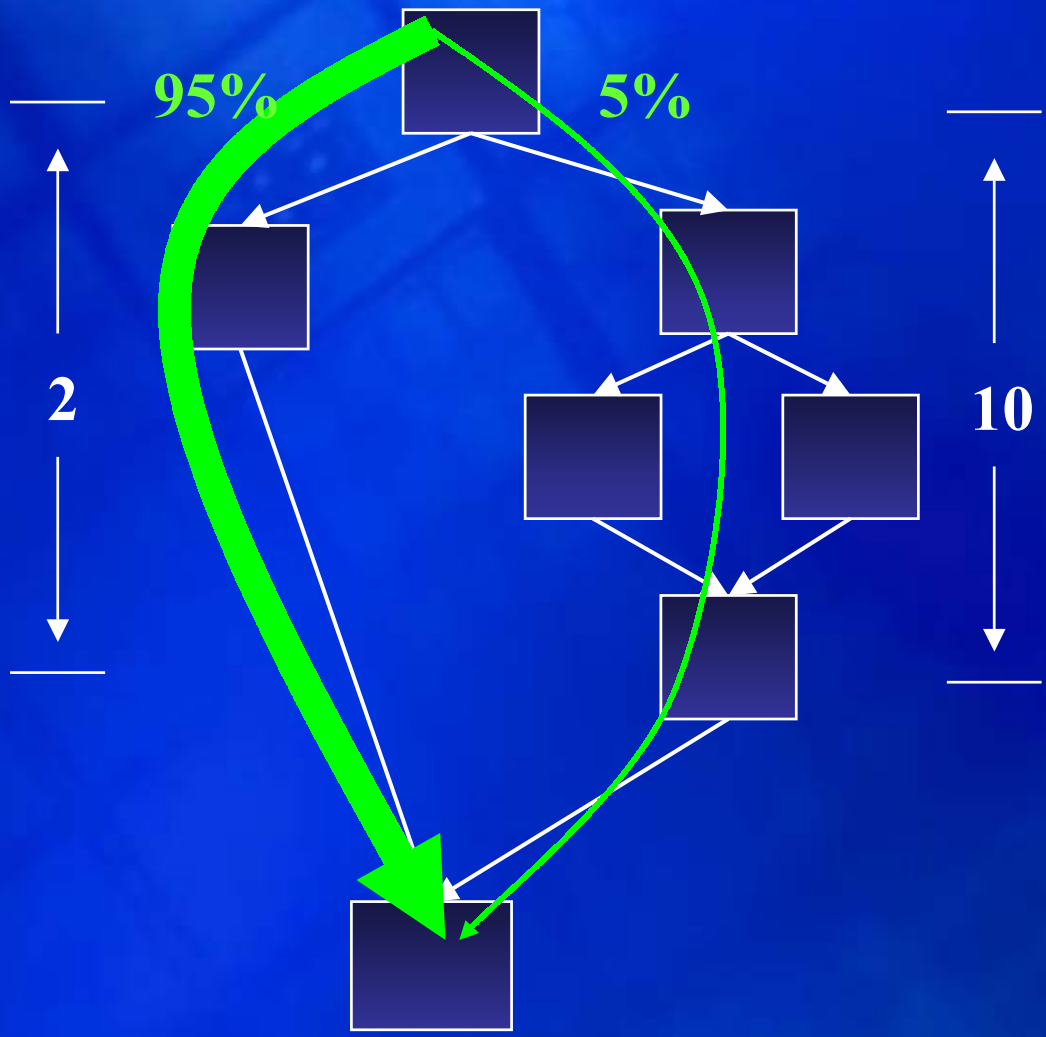


Do we predicate?

- Balanced Paths - Good opportunity to predicate independent of profile

Compilers

Predication

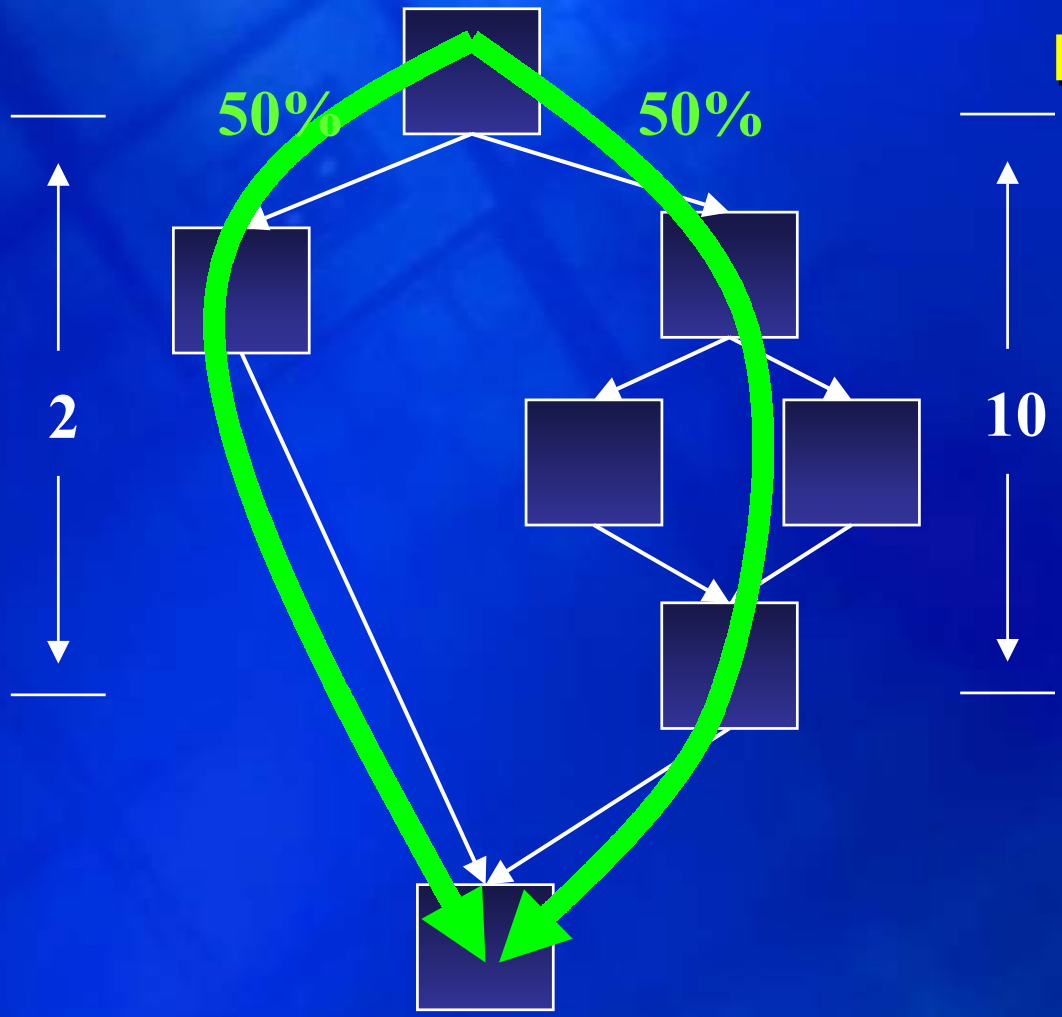


Do we predicate?

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

Compilers

Predication

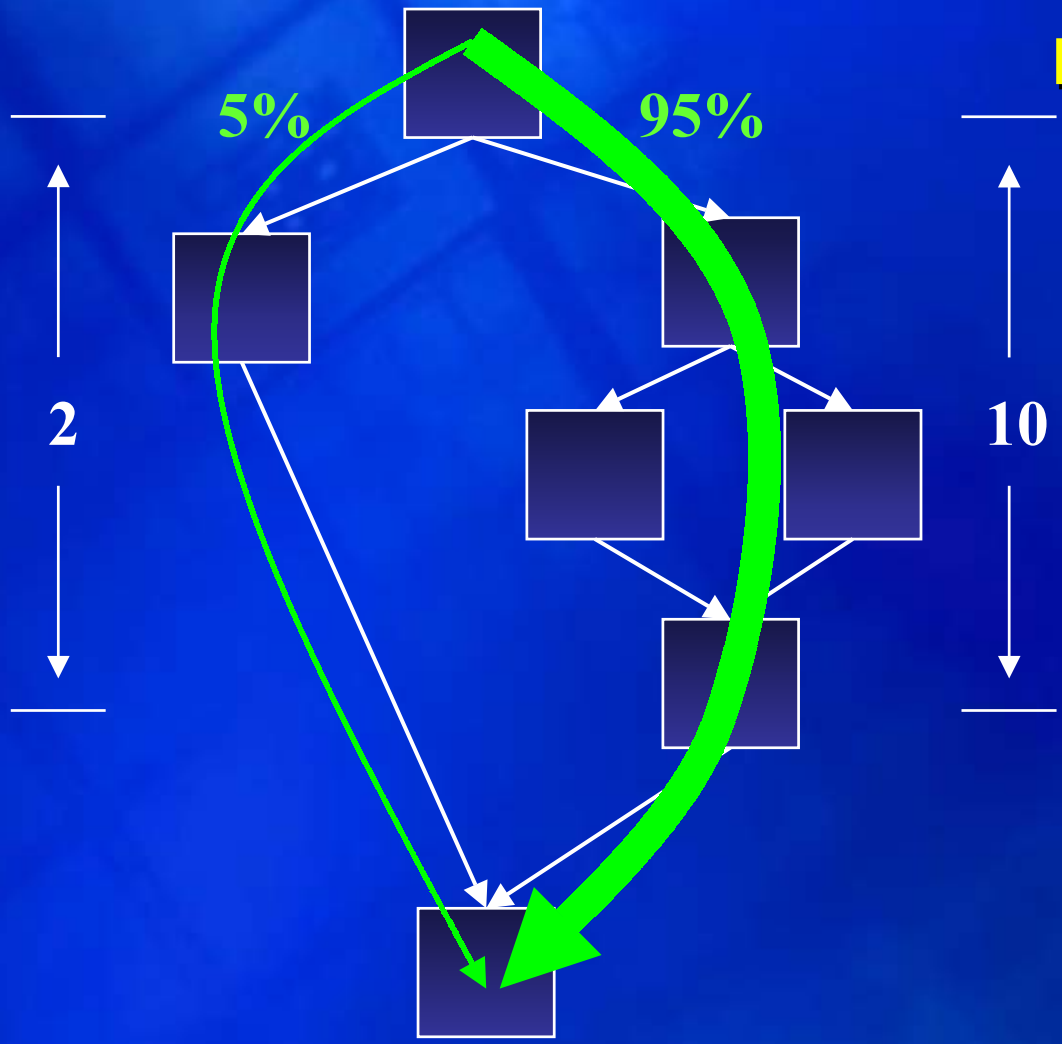


Do we predicate?

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

Compilers

Predication

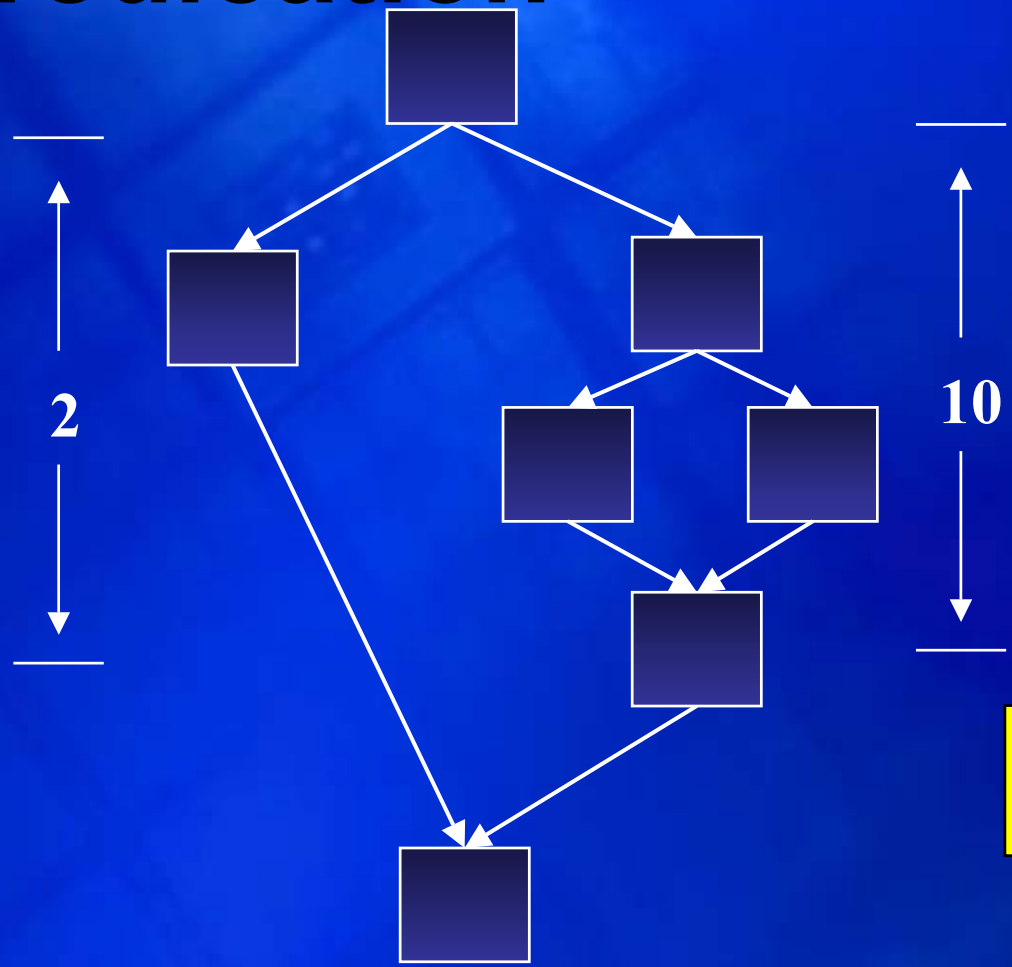


Do we predicate?

Good move.
Left side will slide in for free.

Compilers

Predication



Without profile we won't predicate.

Not any worse than traditional architectures.

Forfeit chance to improve performance.

dgefa with OpenMP* Directives

```
#pragma omp parallel for private(t)
for (j = kp1; j < n; j++) {
   t = a[lda*j+l];
   if (l != k) {
      a[lda*j+l] = a[lda*j+k];
      a[lda*j+k] = t;
   }
   daxpy(n-(k+1),t,&a[lda*k+k+1],1,
      &a[lda*j+k+1],1);
}
```

Optimizing Parallel Loop

```
#pragma omp parallel for private(t) if (n-kp1 >
   40)
for (j = kp1; j < n; j++) {
   t = a[lda*j+l];
   if (l != k) {
      a[lda*j+l] = a[lda*j+k];
      a[lda*j+k] = t;
   }
   daxpy(n-(k+1),t,&a[lda*k+k+1],1,
      &a[lda*j+k+1],1);
}
```