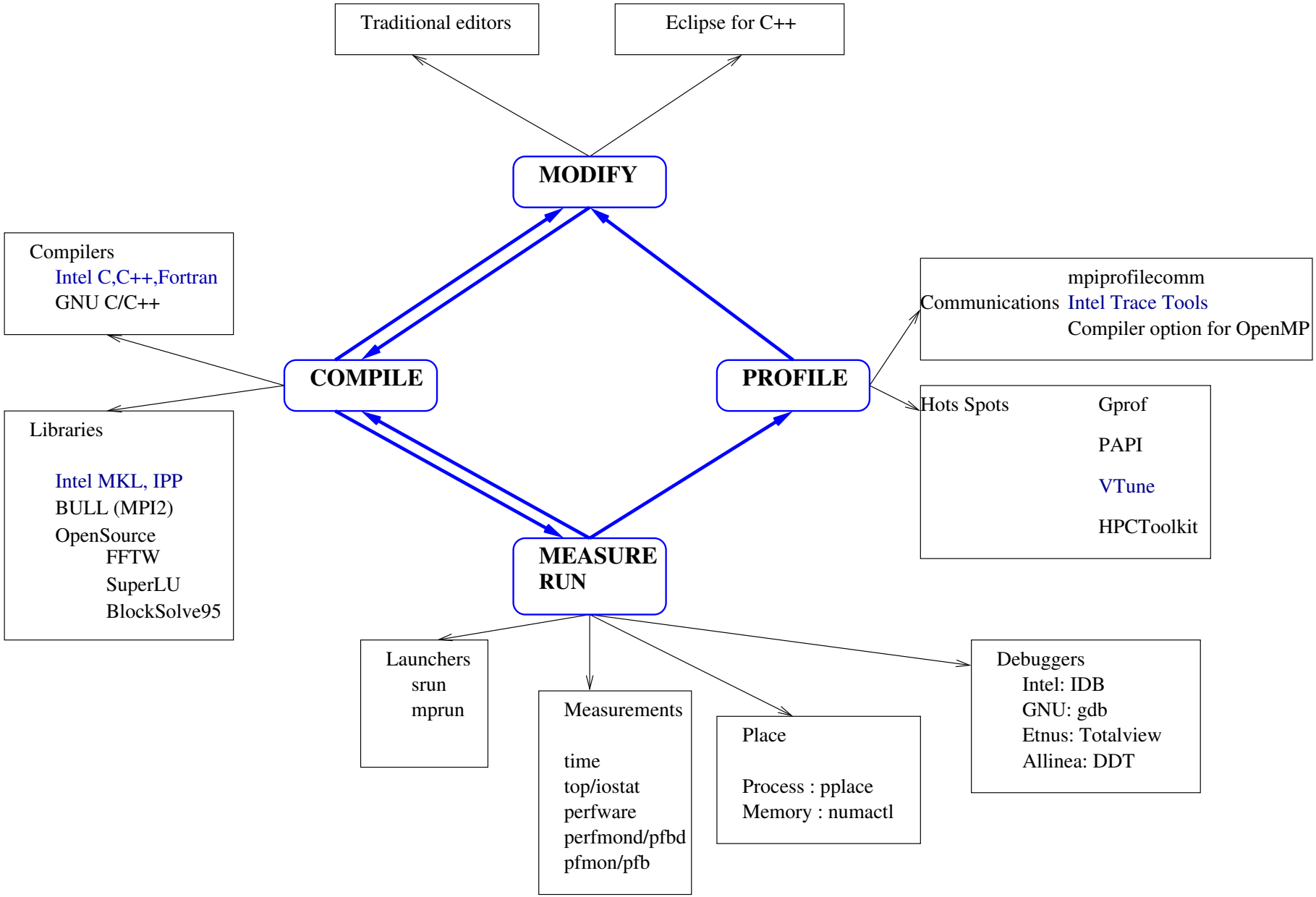
Performance HPC Linux
Bull Echirolles

Software Development Environment

(Part One)

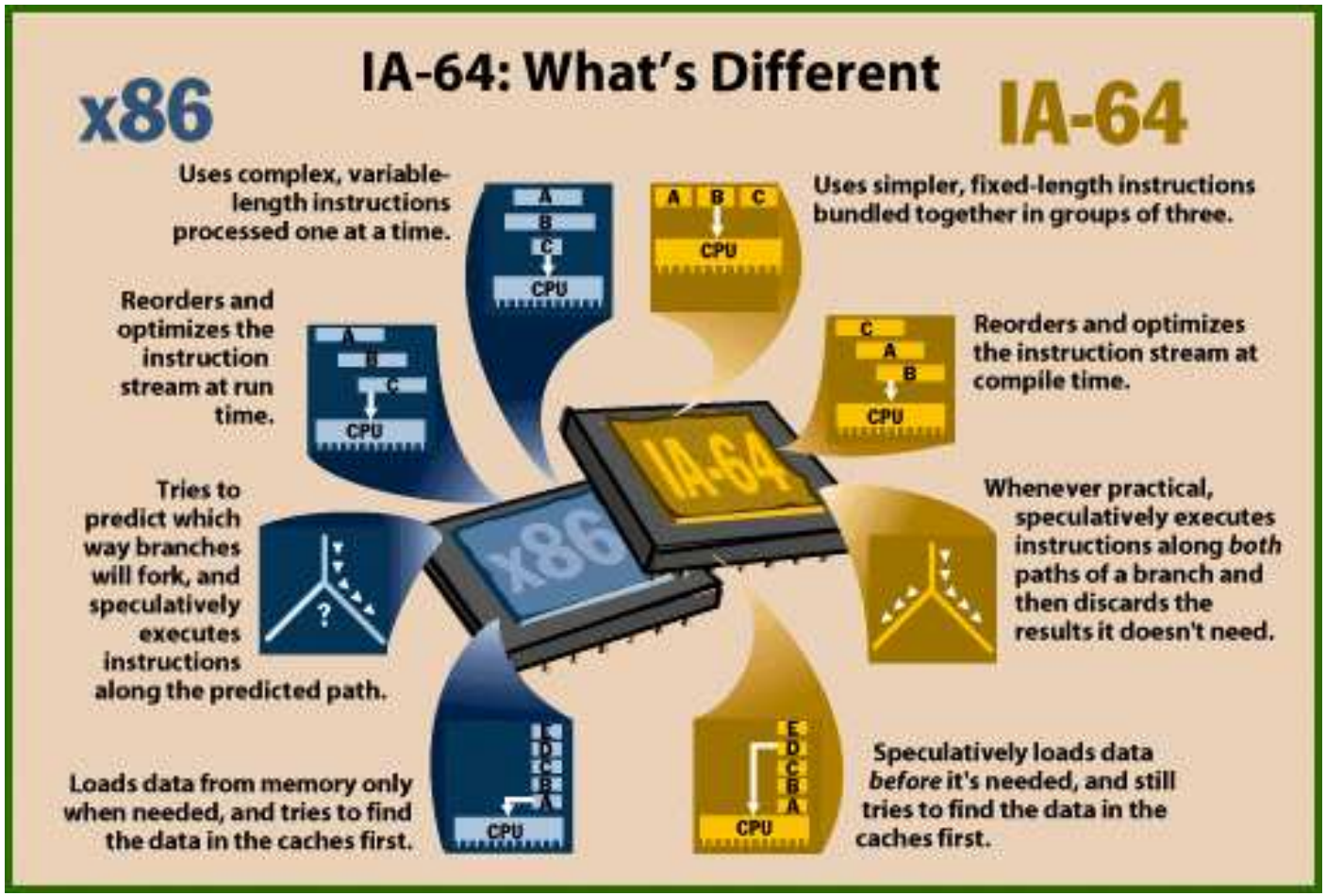
C.Berthelot
Christophe.Berthelot@bull.net


Traditional editors
Eclipse for C++
MODIFY
Compilers
Intel C,C++,Fortran
GNU C/C++
COMPILE
Libraries
Intel MKL, IPP
BULL (MPI2)
OpenSource
FFTW
SuperLU
BlockSolve95
PROFILE
Communications
mpiprofilecomm
Intel Trace Tools
Compiler option for OpenMP
Hots Spots
Gprof
PAPI
VTune
HPCToolkit
MEASURE
RUN
Launchers
srun
mprun
Measurements
time
top/iostat
perfware
perfmond/pfbd
pfmon/pfb
Place
Process : pplace
Memory : numactl
Debuggers
Intel: IDB
GNU: gdb
Etnus: Totalview
Allinea: DDT

- Introduction
- About Intel Compilers
- Help to port
- Optimization
- MPIBull2

- Introduction
  - IA32 vs IA64
  - Glossary
  - Loop optimization
- About Intel Compilers
- Help to port
- Optimization
- MPIBull2

IA32 vs IA64



- Introduction
  - IA32 vs IA64
  - Glossary
  - Loop optimization
- About Intel Compilers
- Help to port
- Optimization
- MPIBull2

Dead Code Elimination

Code that is unreachable or that does not affect the program (e.g. dead stores) can be eliminated.

- Example: In the example below, the value assigned to i is never used, and the dead store can be eliminated. The first assignment to global is dead, and the third assignment to global is unreachable; both can be eliminated.

```
int global;
void f ()
{ int i;
  i = 1;          /* dead store */
  global = 1;     /* dead store */
  global = 2;
  return;
  global = 3;     /* unreachable */}
```

Below is the code fragment after dead code elimination.

```
int global;
void f ()
{ global = 2;
  return;}
```

Loop Unrolling

Loop overhead can be reduced by reducing the number of iterations and replicating the body of the loop.

- Example: In the code fragment below, the body of the loop can be replicated once and the number of iterations can be reduced from 100 to 50.

```
for (i = 0; i < 100; i++)
  g ();
```

Below is the code fragment after loop unrolling.

```
for (i = 0; i < 100; i += 2)
{ g (); g ();}
```

Function Inlining

The overhead associated with calling and returning from a function can be eliminated by expanding the body of the function inline, and additional opportunities for optimization may be exposed as well.

- Example: In the code fragment below, the function add() can be expanded inline at the call site in the function sub().

```
int add (int x, int y)
{return x + y;}
int sub (int x, int y)
{  return add (x, -y);}
```

Expanding add() at the call site in sub() yields:

```
int sub (int x, int y)
{  return x + -y;}
```

Loop Fusion

Some adjacent loops can be fused into one loop to reduce loop overhead and improve run-time performance.

- Example: The two adjacent loops on the code fragment below can be fused into on loop.

```
for (i = 0; i < 300; i++)
  a[i] = a[i] + 3;
for (i = 0; i < 300; i++)
  b[i] = b[i] + 4;
```

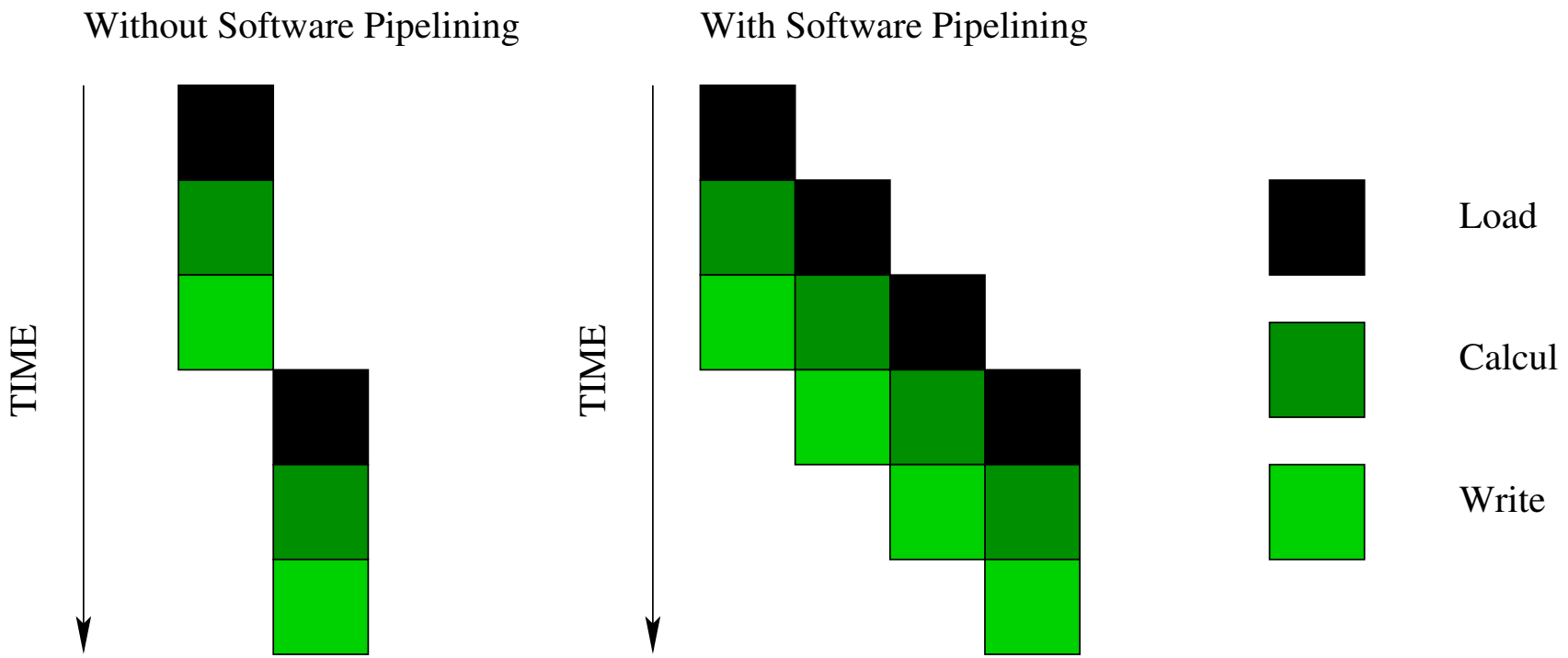
Below is the code fragment after loop fusion.

```
for (i = 0; i < 300; i++)
  {
    a[i] = a[i] + 3;
    b[i] = b[i] + 4;
  }
```

Aliasing

Two variables are aliased if they can refer to the same storage location.

Pipelining



- Introduction
  - IA32 vs IA64
  - Glossary
  - Loop optimization
- About Intel Compilers
- Help to port
- Optimization
- MPIBull2

Switching

Switching, if possible, within loops is useful to align the access to arrays with their position in memory.

```
do i = 1, N
   do j = 1, N
      A(i,j) = 1/B(i,j)
   end do
end do
```

real 73.13s

```
do j = 1, N
   do i = 1, N
      A(i,j) = 1/B(i,j)
   end do
end do
```

real 39.09s

Loop tiling or loop blocking

The partitioning of loops allows their granularity to be adapted to the memory hierarchy. The computation is done by blocs which are not necessarily aligned. This works well when all the loops may be switched.

```
do jj = 1, N, sj
  do ii = 1, N, si
    do j = jj, jj+sj-1
      do i = ii,ii+si-1
        A(i,j) = 1/B(i,j)
      end do
    end do
  end do
end do
```

real 41.68

Loop Peeling

Loop splitting (or loop peeling) is a compiler optimization technique. It attempts to simplify a loop or eliminate dependencies by breaking it into multiple loops which have the same bodies but iterate over different contiguous portions of the index range.

```
do i = 1, N
   A(i) = A(1) + B(i);
end do
```

```
A(1) = A(1) + B(1);
do i = 2, N
   A(i) = A(1) + B(i);
end do
```

- Introduction
- About Intel Compilers
  - Default configuration
  - KMP variables
- Help to port
- Optimization
- MPIBull2

Intel Compilers

Compilers play an essential role in exploiting the full potential of Itanium®2 processors. The main features of this compiler are:

- Optimization of throughput of floating point instructions
- Optimization of inter-process calls
- Data preloading
- Conditional instruction prediction
- Speculative loading
- Optimization of the software pipeline
- Support OpenMP
- Compatible with GNU products.

Default configuration

Option by default

The configuration file to use instead of the default configuration file:

- Variable IFORTCFG for fortran
- Variable ICCCFG for C
- Variable ICPCCFG for C++

Version

To use C/C++ and Fortran you must have the same Build :

```
Intel(R) Fortran IA-64 Compiler for applications running on IA-64, Version 10.0
Build 20080312 Package ID: l_fc_p_10.1.015
Copyright (C) 1985-2007 Intel Corporation.  All rights reserved.

Intel(R) C IA-64 Compiler for applications running on IA-64, Version 10.0
Build 20080312 Package ID: l_cc_p_10.1.015
Copyright (C) 1985-2007 Intel Corporation.  All rights reserved.
```

KMP variables: OpenMP impact

- KMP_LIBRARY=a,
  - a can be serial,turnaround, throughput indicating the execution mode
- KMP_STACKSIZE Sets the number of bytes to allocate for each parallel thread to use as its private stack (the default onItanium(R)-based systems is 4m).
- KMP_AFFINITY (with 10.0 compiler)
  - granularity=fine,compact: Specifying compact binds the OpenMP thread <n>+1 on a free thread context as close as possible to the thread context where the <n> OpenMP thread was bound.
  - granularity=fine,scatter: Specifying scatter distributes the threads as evenly as possible across the entire system.

- Introduction
- About Intel Compilers
- Help to port
  - Option to port
  - Option to report
  - Memory problems
- Optimization
- MPIBull2

Options to port 1/2

Fortran

- `-warn all` Enables all warning messages (compile time).
- `-warn errors` Change all warning-level messages into error-level messages
- `-check all` Enables all check options (Runtime)
- `-fpe 0` This setting provides full IEEE support.

C/C++

- `-O0` Disables all optimizations
- `-w2` Enables all warning messages.
- `-Werror` Change all warning-level messages into error-level messages
- `-Wp64` Enables 64-bit porting specific warnings

Options to port 2/2

Both

- `-fmath-errno` Tells the compiler to assume that the program tests errno after calls to math library functions
- `-mp` Maintain floating-point precision
- `-dryrun` Tells the driver that tool commands should be shown but not executed.

Example with Wp64

```
int main() {
long anumber = 5;
int number;
number = anumber;
return 1;}
```

Use icc -Wp64

```
test2.cpp(8): warning #810: conversion from "long" to "int" may
              lose significant bits
  number = anumber;
         ^
```

Options to report

- `-opt_report` : Tells the compiler to generate an optimization report to stderr.
- `-opt_report_file`*file.txt* : Specifies the name for an optimization report
- `-opt_report_level`{*min*|*med*|*max*} : Specifies the detail level of the optimization report.
- `-opt_report_phase`*phase* : Specifies the optimizer phase to use when reports are generated.
  - `ecg_swp` Code Generator/software pipelining
  - `hlo` High Level Optimizer
  - `ipo` Interprocedural Optimizer
- `-opt_report_help` : Displays the optimizer phases available for report generation.

Example on loop skewing

Before loop skewing

```
do i=0,N                                  Swp report for loop at line 16 in MAIN_ in file Skewing.f90
   do j=0,i+2
      A(i,j)=A(i-1, j) + A(i, j-1)        According to the estimate of the Modulo Scheduler, the acyclic
   end do                                 global scheduler can achieve a better schedule than software
end do                                    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    17 --> Load   at line    17
```

After loop skewing

```
do t=0,2*N+2                              Swp report for loop at line 16 in MAIN_ in file Skewing2.f90
 do p=MAX(0,t-N) , MIN(t,t/2+1)
   A(t-p, p)=A(t-p-1, p)+A(t-p, p-1)   Number of stages in the software pipeline =    4
 end do
end do
```

Fortran check option

Check option

- Compile with -traceback -g
- -check bounds : Performs run-time checks on whether array subscript and substring references are within declared bounds.

Example

```
1 program test
2    implicit none
3    real(kind=8) ,allocatable, dimension(:) :: tab1
4    allocate(tab1(10000))
5    tab1(10001)=1.0
6    print *,"Tab(10000)=", tab1(10000)
7 end program test
```

With -check all

```
forrtl: severe (408): fort: (2):
Subscript #1 of the array TAB1 has value 10001 which is greater than the upper bound of 10000
Image              PC                Routine            Line        Source
test_elec          400000000000A9690  Unknown               Unknown  Unknown
test_elec          400000000000A5B20  Unknown               Unknown  Unknown
test_elec          40000000000463A0  Unknown               Unknown  Unknown
test_elec          40000000000049C0  Unknown               Unknown  Unknown
test_elec          4000000000004C90  Unknown               Unknown  Unknown
test_elec          4000000000003050  MAIN__                     5  test2.f90
test_elec          4000000000002A00  Unknown               Unknown  Unknown
libc.so.6.1        2000000000469430  Unknown               Unknown  Unknown
test_elec          4000000000002780  Unknown               Unknown  Unknown
```

Use ElectricFence

ElectricFence

- malloc() debugger for Linux and Unix. This will stop your program on the exact instruction that overruns or under-runs a malloc() buffer.
- Compile with -g -traceback and link with -lefence
- Or use LD_PRELOAD=/usr/lib/libefence.so (inside script)

Example

```
1 program test
2    implicit none
3    real(kind=8) ,allocatable, dimension(:) :: tab1
4    allocate(tab1(10000))
5    tab1(10001)=1.0
6    print *,"Tab(10000)=", tab1(10000)
7 end program test
```

ElectricFence

- Compile with -g and -traceback -lefence
- Results

```
 Electric Fence 2.2.0 Copyright (C) 1987-1999 Bruce Perens <bruce@perens.com>
forrtl: severe (174): SIGSEGV, segmentation fault occurred
Image              PC                Routine            Line        Source
test_elec          4000000000002D81  MAIN__                     5  test2.f90
```

Use Dmalloc

Dmalloc

- Link with -ldmalloc
- Use DMALLOC_OPTIONS
  - turn on transaction and stats logging and set 'logfile' as the log-file DMALLOC_OPTIONS=log-trans,log-stats,log=logfile
  - enable 'logfile' as the log-file, watch for address '0x1234', and start checking when we see file.c line 123: DMALLOC_OPTIONS log=logfile,addr=0x1234,start=file.c:123

Example 2/3

```
1 program test
2    implicit none
3    real(kind=8) ,allocatable, dimension(:) :: tab1
4    allocate(tab1(10000))
5    tab1(10001)=1.0
6    print *,"Tab(10000)=", tab1(10000)
7 end program test
```

Dmalloc

- Compile with -g and -traceback -ldmalloc

- Introduction
- About Intel Compilers
- Help to port
- Optimization
  - Simples options
  - Inlining
  - Unrolling
  - Floating Point
  - Pointer
  - Profile Guided
  - Interprocedural
  - Control memory bandwidth
  - OpenMP
  - linker
- MPIBull2

Three levels for optimization

- -01 Optimize for code size
- -02 Default option enables optimizations for speed, including global code scheduling, software pipelining, predication, and speculation
- -03 Advanced optimization. Enables 02 optimizations plus more aggressive optimizations

Control inlining

Inlining

- `-ip` : Enables interprocedural optimizations for single file compilation.
- `-Qoption,f/c,-ip_ninl_min_stats=n` : Sets the valid maximum number of intermediate language statements for a function that is expanded in line
- `-Qoption,f/c,-ip_ninl_max_total_stats=n` : Sets the maximum increase in size of a function, measured in intermediate language statements, due to inlining

Control Unrolling

Unrolling

- `-unroll0` : disable loop unrolling,
- `-unroll` : Enables loop unrolling,
- `-unrollM` : Sets the maximum number of times to unroll loops

Control Floating Point

- -IPF-fp-relaxed : Enables use of faster but slightly less accurate code sequences for math functions, such as divide and sqrt.
- -IPF-fp-speculation<mode> : Tells the compiler to speculate on floating-point (FP) operations in one of the following
  - fast - Speculate on floating-point operations. This is the default.
  - safe - Speculate on floating-point operations only when safe.
  - strict - This is the same as specifying off.
  - off - Disables speculation of floating-point operations.

Pointers

- -fno-alias: all pointers are assumed not to alias
- -fno-fnalias: assume no aliasing within function (pointer arguments are unique)
- -[no]restrict Enable [disable] the "restrict" keyword for disambiguating pointers
- -ipo: global analysis can disambiguate pointer
- -ivdep-parallel Tells the compiler that there is no loop-carried memory dependency in any loop following an IVDEP directive.

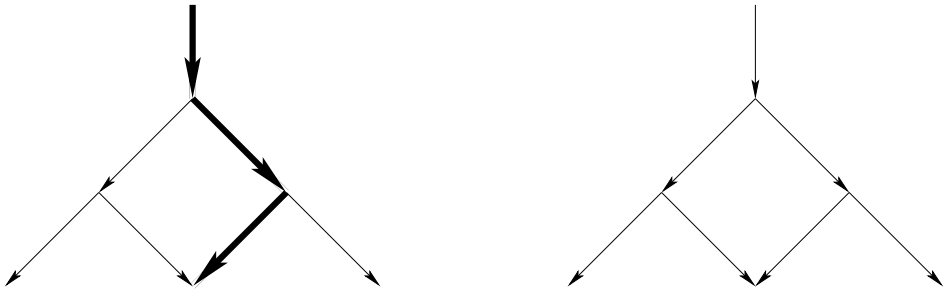
Profile Guided

Impacts

- Dynamic branch prediction, Loop, Cache utilization
- Speculation, Function splitting

Benefict with

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



Profile Guided: how to use

Compilation

- Compile with -prof_gen $\longrightarrow$ instrumented binary
- Run it $\longrightarrow$ file .dpi
- Compile with -prof_use + dpi file $\longrightarrow$ optimized binary

Example with povray and woodbox.pov example

- Without option : 23.0s
- With PGO : 20.0s (13%)

Interprocedural

Impacts

- Inlining
- Information propagation (Alignment)
- Whole program optimization

Interprocedural Optimizations: how to use

Compilation

- Compile with -ipo → obj
- Link with -ipo → optimized binary
- Use xiar to create a library
- If you use ld change to xild

Example with povray and woodbox.pov example

- Without option : 23.0s
- With PGO : 19s (17%)

Memory bandwidth

- `-opt-mem-bandwidth<n>` to control memory bandwidth
  - 0 Enables a set of performance tuning and heuristics in compiler optimizations that is optimal for serial code. This is the default for serial code.
  - 1 Enables a set of performance tuning and heuristics in compiler optimizations for multithreaded code generated by the compiler. This is the default if compiler option -parallel or -openmp is specified
  - 2 Enables a set of performance tuning and heuristics in compiler optimizations for parallel code such as Windows Threads, pthreads, and MPI code, besides multithreaded code generated by the compiler.

OpenMP

Compile

- To use openmp `-openmp`
- Enables analysis of OpenMP applications : `-openmp_profile`

Example CG from NAS

```
Region counts:
  serial regions   : 19
  barrier regions  : 8
  parallel regions : 15
end
Program execution time (in seconds):
  cpu              :     0.11 sec
  elapsed          :    71.16 sec
    serial         :     0.16 sec
    parallel       :    71.00 sec
  cpu percent      :     0.16 %
end
```

Directives

- Before the loop : `#pragma` for C/C++. `[Cc*!]DIR$` for fortran
- Directives before a loop
  - `[NO]SWP` directive enables software pipelining
  - `LOOP COUNT(N)` Specifies the loop count this assists the optimizer.
  - `[NO]UNROLL, UNROLL(N)` Tells the compiler's optimizer how many times to unroll a DO loop or disables the unrolling
  - `[NO]PREFETCH a:b:c` Enables or disables a data prefetch from memory.
  - `IVDEP` Assists the compiler's dependence analysis
  - `MEMORYTOUCH` directive allows the programmer to inform the processor that it will be reading or writing a memory range in the future
  - `OPTIMIZE` General Compiler Directive: Enables or disables optimizations.

Example: with SWP

Without directive

```
do i = 1, m                    Resource II   =        1
 if (a(i) .eq. 0) then         Recurrence II =        1
    b(i) = a(i) + 1            Minimum II    =        1
 else                          Last attempted II  =   1
    b(i) = a(i)/c(i)           Estimated GCS II   =   1
 endif                         Modulo scheduling was successful,but there was no overlap
enddo                          across iterations => loop not pipelined
```

With SWP directive

```
!DIR$ SWP
do i = 1, m                    Resource II   =        1
 if (a(i) .eq. 0) then         Recurrence II =        1
    b(i) = a(i) + 1            Minimum II    =        1
 else                          Scheduled II  =        1
    b(i) = a(i)/c(i)           Estimated GCS II   =   1
 endif                         Percent of Resource II needed by arithmetic ops     = 100%
enddo                          Percent of Resource II needed bymemory ops         = 100%
                               Percent of Resource II needed by floating point ops =   0%
                               Number of stages in the software pipeline =    1
```

Loop interchange

- Use `-O3 -opt_report -opt_report_level max -opt_report_phase hlo`
- Problem
  - Interchange not done due to function call inside
  - Interchange not done due to imperfect loopnest
  - Interchange not done due to data dependencies: dependencies preventing interchange are also reported
  - interchange not done when the original order was found to be proper, but there were some close calls!

Example: Function Call Inside loop

```
void bar (int *A, int **B);
int foo (int *A, int **B, int N)
{
   int i, j;
   for (j=0; j<N; j++) {
      for (i=0; i<N; i++) {
         B[i][j] += A[j];
         bar(A,B);
      }
   }
   return 1;
}
```

```
HLO Report:
<test.c;8:8;hlo_linear_trans;foo;0>
Loop Interchange Not Done due to: User Function Inside Loop Nest
Advice: Loop Interchange, if possible, might help Loopnest at lines: 8
10
          : Suggested Permutation: (1 2 ) --> ( 2 1 )
```

Linker option

Multiple

- To compile with multiple def: -Wl,-z,muldefs
- To have informations: -Wl,-z,muldefs,-M

Example 1/2

File lib1.f90->inside libtest1.a

```
subroutine test ()
     implicit none
     print *, "Inside  lib1"
end subroutine test
```

File test2.f90->inside libtest2.a

```
subroutine test ()
     implicit none
     print *, "Inside  lib2"
end subroutine test
```

File pprincipal.f90

```
program pprincipal
     implicit none
     call test()
end program pprincipal
```

Example 2/2

Compile

- Use :

```
ifort  -Wl,-z,muldefs -Wl,-M    -o pprincipal pprincipal.o -L. -ltest1 -ltest2
```

- We have

```
fonction test
./libtest1.a(lib1.o)          pprincipal.o (test_)
```

- Introduction
- About Intel Compilers
- Help to port
- Optimization
- MPIBull2
  Environment Variables

Mpibull2: Astlik

The Armenian goddess of love and fertility. With the sun god Vahagn and the moon goddess Anahit she forms an astral trinity. She is similar to the Greek Aphrodite and the Mesopotamian Ishtar. Her name means "little star".

MPIBull2

MPIBull2 is based on Argonne National Laboratory's MPICH-2 implementation of the MPI-2 specification. It Supports MPI 1.2 and MPI 2 Standard functionality:

- Supports both MPI 1.2 and MPI 2 standard functionalities including
  - Dynamic processes (osock only)
  - One-sided communications
  - Extended collectives
  - Thread safety (osock, oshm and elanbull2)
  - Latest ROMIO including the latest patches developed by Bull
- Multi-device functionality:
  - Sockets-based messaging (for Ethernet, SDP, SCI and EIP)
  - Hybrid shared memory-based messaging for shared memory
  - Quadrics network drivers (qxelan, elanbull2)
  - InfiniBand architecture multirails driver Gen2

Environment Variables

To compile

- MPIBULL2_PRELIBS
- MPIBULL2_POSTLIBS

To run

All variables : mpibull2-params -lall

- MPI_QUIET

(To be continued)

Architect of an Open World™

- (c) Copyright Bull. All rights reserved
  - ✔ Users Restricted Rights - Use, duplication or disclosure restricted.
  - ✔ Any copy of these documents should keep all copyright, logos and other proprietary notices contained herein.
  - ✔ This publication may include technical inaccuracies or typographical errors.
  - ✔ This publication is provided "AS IS" without any warranty either expressed or implied including but not limited to the implied warranties of merchantabilities or fitness of the described product.
  - ✔ Course Material Licensing Terms : No sublicensing rights.
  - ✔ For other licensing needs, please contact Bull