Performance HPC Linux
Bull Echirolles

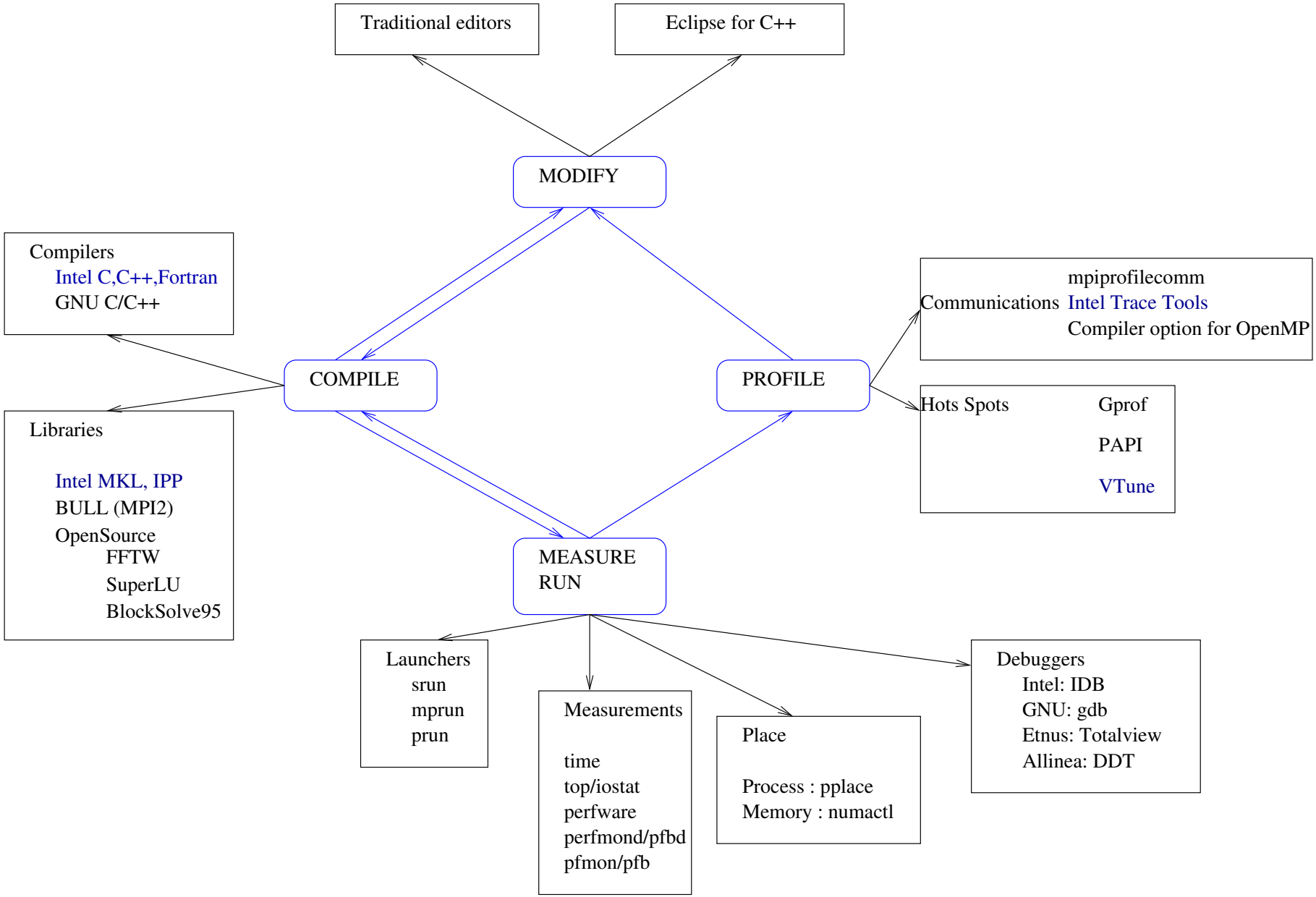
Software Development Environment

(Part One)

C.Berthelot
Christophe.Berthelot@bull.net

Copyright (©) Bull S.A.S. 2007-2009

© Bull 2009


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
MEASURE
RUN
Launchers
srun
mprun
prun
Measurements
time
top/iostat
perfware
perfmond/pfbd
pfmon/pfb
Place
Process : pplace
Memory : numactl
Debuggers
Intel: IDB
GNU: gdb
Etnus: Totalview
Allinea: DDT

© Bull 2009

- Introduction
- About Intel Compilers
- Help to port
- Optimization
- Bug Report
- MPIBull2
- Profiling inside the node
- Profiling : MPI

© Bull 2009

- Introduction
  - Glossary
  - Loop optimization
- About Intel Compilers
- Help to port
- Optimization
- Bug Report
- MPIBull2
- Profiling inside the node
- Profiling : MPI

© Bull 2009

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;}
```


© Bull 2009

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 ();}
```


© Bull 2009

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;}
```


© Bull 2009

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;
  }
```

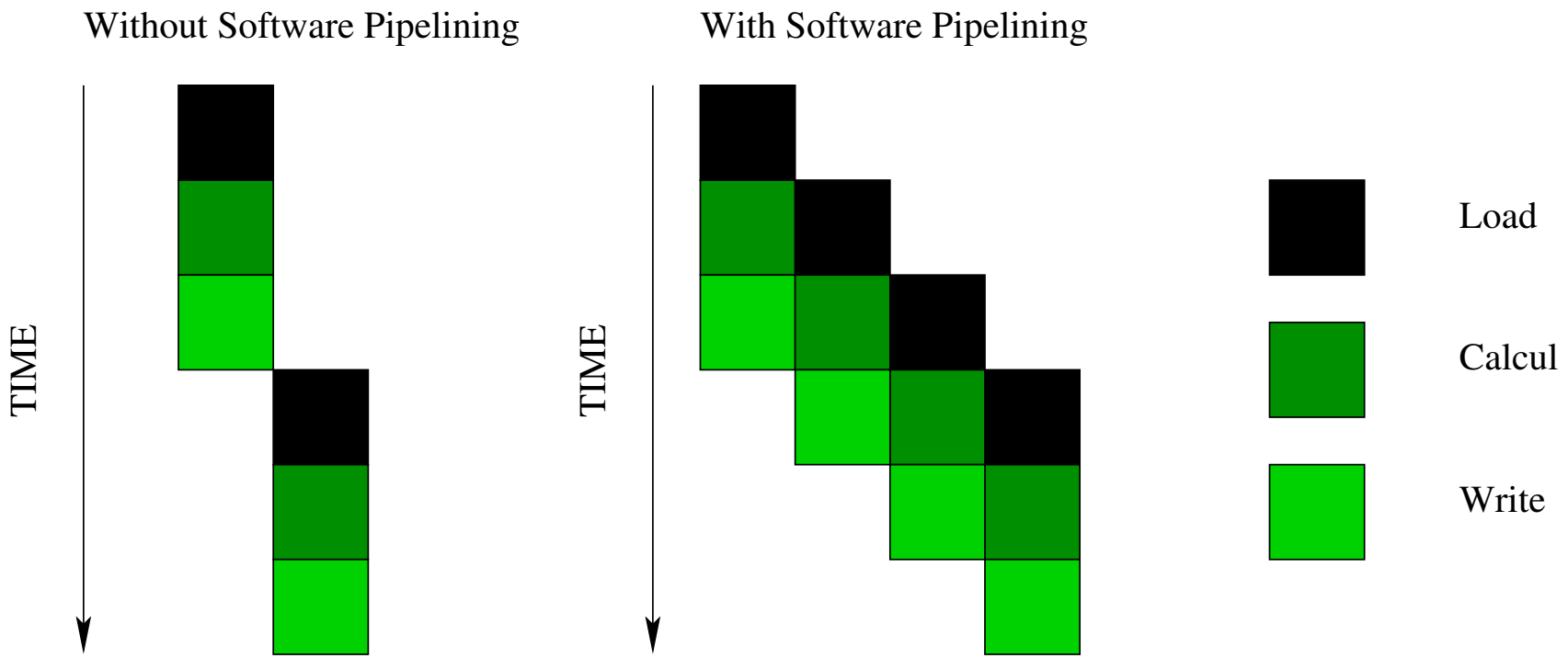

© Bull 2009

Aliasing

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

© Bull 2009

Pipelining



© Bull 2009

- Introduction
  - Glossary
  - Loop optimization
- About Intel Compilers
- Help to port
- Optimization
- Bug Report
- MPIBull2
- Profiling inside the node
- Profiling : MPI

© Bull 2009

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

© Bull 2009

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

© Bull 2009

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
```


© Bull 2009

Loop skewing

Loop skewing takes a nested loop iterating over a multidimensional array, where each iteration of the inner loop depends on previous iterations, and rearranges its array accesses so that the only dependencies are between iterations of the outer loop.

```
do i=0,N
  do j=0,i+2
    A(i,j)=A(i-1, j) + A(i, j-1)
  end do
end do
```

```
do t=0,2*N+2
  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)
  end do
end do
```


© Bull 2009

- Introduction
- About Intel Compilers
  - Default configuration
  - KMP variables
- Help to port
- Optimization
- Bug Report
- MPIBull2
- Profiling inside the node
- Profiling : MPI

© Bull 2009

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.

© Bull 2009

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 20070426 Package ID: l_fc_p_10.0.023
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 20070426 Package ID: l_cc_p_10.0.023
Copyright (C) 1985-2007 Intel Corporation.  All rights reserved.
```


© Bull 2009

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=compact,<level> (with 10.0 compiler)

© Bull 2009

- Introduction
- About Intel Compilers
- Help to port
  - Option to port
  - Option to report
  - Memory problems
- Optimization
- Bug Report
- MPIBull2
- Profiling inside the node
- Profiling : MPI

© Bull 2009

Options to port 1/2

Fortran

- `-warn all` Enables all warning messages (compile time).
- `-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.
- `-Wp64` Enables 64-bit porting specific warnings

© Bull 2009

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.

© Bull 2009

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;
         ^
```


© Bull 2009

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.

© Bull 2009

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
```


© Bull 2009

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.

© Bull 2009

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
```


© Bull 2009

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)

© Bull 2009

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
```


© Bull 2009

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

© Bull 2009

- Introduction
- About Intel Compilers
- Help to port
- Optimization
  - Simples options
  - Inlining
  - Unrolling
  - Floating Point
  - Pointer
  - Profile Guided
  - Interprocedural
  - Control memory bandwidth
  - OpenMP
  - linker
- Bug Report
- MPIBull2
- Profiling inside the node
- Profiling : MPI

© Bull 2009

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

© Bull 2009

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

© Bull 2009

Control Unrolling

Unrolling

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

© Bull 2009

Control Floating Point

- -fp-relaxed : Enables use of faster but slightly less accurate code sequences for math functions, such as divide and sqrt.
- -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.

© Bull 2009

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.

© Bull 2009

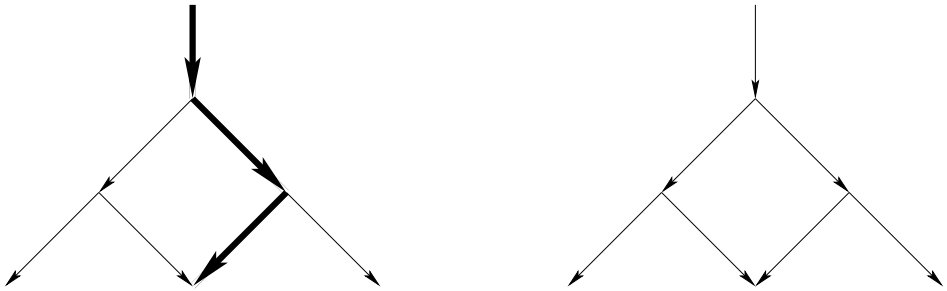
Profile Guided

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



© Bull 2009

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%)

© Bull 2009

Interprocedural

Impacts

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

© Bull 2009

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%)

© Bull 2009

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.

© Bull 2009

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
```


© Bull 2009

Directives

- Before the loop : `#pragma` for C/C++. `[Cc*!]DIR$` for fortran
- Directives before a loop
  - `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
  - `VECTOR ALWAYS` enables or disables vectorization of a DO loop.
  - `OPTIMIZE` General Compiler Directive: Enables or disables optimizations.

© Bull 2009

Example: with VECTOR ALWAYS

Without directive

```
do t=0,2*N+2                              16:1:VEC:skew2_: loop was not vectorized: not inner loop
 do p=MAX(0,t-N) , MIN(t,t/2+1)           17:4:VEC:skew2_: loop was not vectorized: vectorization
   A(t-p, p)=A(t-p-1, p)+A(t-p, p-1)      possible but seems inefficient
 end do
end do
```

With VECTOR ALWAYS directive

```
do t=0,2*N+2                              16:1:VEC:skew_:loop was not vectorized: not inner loop
!DEC$ VECTOR ALWAYS
 do p=MAX(0,t-N) , MIN(t,t/2+1)           18:4:VEC:skew:  LOOP WAS VECTORIZED
   A(t-p, p)=A(t-p-1, p)+A(t-p, p-1)
 end do
end do
```


© Bull 2009

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!

© Bull 2009

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 )
```


© Bull 2009

Linker option

Multiple

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

© Bull 2009

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
```


© Bull 2009

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_)
```


© Bull 2009

- Introduction
- About Intel Compilers
- Help to port
- Optimization
- Bug Report
  «Brutus» options
  Tool: CESR
- MPIBull2
- Profiling inside the node
- Profiling : MPI

© Bull 2009

Brutus options

Applicability

Program builds and runs correctly without optimization, but fails with optimization. A compiler bug is suspected

Method

- "good" and "bad" set of switches : -O2 ok -O3 ko
- Identify source file responsible for error
- *Identify function within source file*
- *Identify compiler optimization responsible*

© Bull 2009

Identify function within source file

Bad options plus from_rtn 0 and to_rtn m

- First compile with bad options and from_rtn 0 to_rtn 1000 to get number of functions/subroutines in file
- Compile with -from_rtn 0 -to_rtn m. Goal find lowest value of m to give error: ifort -from_rtn 3 -to_rtn 5 -g -c O2 -FR raflib.f

```
Not optimizing (phase 2) module_pmat1$pro333_ 1
Not optimizing (phase 2) module_pmat1$dpro333_ 2
Still optimizing (phase 2) module_pmat1$cro33_ 3
Still optimizing (phase 2) module_pmat1$dcro33_ 4
Still optimizing (phase 2) module_pmat1$norv_ 5
Not optimizing (phase 2) module_pmat1$dnorv_ 6
Not optimizing (phase 2) module_pmat1$norq_ 7
Not optimizing (phase 2) module_pmat1$dnorq_ 8
Not optimizing (phase 2) module_pmat1$swpvv_ 9
```


© Bull 2009

Identify compiler optimization responsible

Add option num_opt

- Find total number of compiler optimizations invoked
- Compile with -from_rtn 0 -to_rtn m -num_opt k. Goal determine the lowest value of k that gives error : ifort from_rtn 1 to_rtn 70 O3 num_opt 300 badfile.f

```
Doing bottom testing 1 (phase 1)
Doing il0 peep 2 (phase 1)
Doing optimize igoto 3 (phase 1)
```


© Bull 2009

Tool: CESR

Compiler Error Source Reduction

Monitor a compiler (or script) to instrument success/failure of numerous runs while reducing a target source file to minimal size.

Example

```
Final reproducing compiler command:
>>> icc  -from_rtn 1 -to_rtn 3  -num_opt 68  -O3  cesrcut.c

Summary Statistics:
    Compiler invocations:            623
    Time (sec) running compiler:     51.315
    Time (sec) total:                95.307
    Lines of code (initial):         3439
    Lines of code (final):           18
```


© Bull 2009

- Introduction
- About Intel Compilers
- Help to port
- Optimization
- Bug Report
- MPIBull2
  - How to start
  - Environment Variables
- Profiling inside the node
- Profiling : MPI

© Bull 2009

Freyja

Freyja is quite independent, being chief of the Valkyries, the demi-goddesses who select the noble and heroic dead and carry them to the Realm of the Gods.Freya is the Daughter of Time.

Aakuluujjusi

Aakuluujjusi created the animals that humans eat. She created the caribou by taking them off her trousers and placing them on the earth. Next Aakuluujjusi placed her jacket on the ground, creating the walrus, then placed antlers on its head. The Intuit people were upset because the two new animals attacked them. So Aakuluujjusi swapped the tusks to the walrus and the antlers to the caribou. The Inuit people still complained that the caribou was too fast, so Aakuluujjusi turned the caribou's belly-hair around so that the hairs caught in the wind, slowing the caribou down.

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".

© Bull 2009

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

© Bull 2009

How to start

First of all

- First time you must source /opt/mpi/mpibull2-<version>/share/setenv_mpibull2
- Use module
  - module use -a /opt/mpi/mpibull2-<version>/share/modules
  - module load mpibull2.modules

© Bull 2009

Commands

- mpibull2-devices -c : Get my configuration
- mpibull2-devices -dl : Get the list of available devices
- mpibull2-devices -d=<device>: Change to this device if available

Scripts

- mpif77, mpif90, mpicc and mpiCC

© Bull 2009

Environment Variables

To compile

- MPIBULL2_PRELIBS
- MPIBULL2_POSTLIBS

To run

- MPI_QUIET
- NUM_RDMA_BUFFER (only for device ibmr_gen2)
- MPIBULL2_USE_POSTAL (default with elanbull2)

© Bull 2009

Tool

Classical open source gprof

Advantages

- Standard
- Well known
- Supported by Intel compiler
- Easy to use
- MPI support

© Bull 2009

Example

Traditional tool: gprof

- Simple to use (compile with -p)
- Use GPROF_PREFIX or GMON_OUT_PREFIX
- gprof -s a.out $GMON_OUT_PREFIX.*
- gprof a.out gmon.sum > summary-profile

© Bull 2009

HPCToolKit with XBAS (Bull Kernel)

HPCToolkit is an open-source suite of multi-platform tools for profile-based performance analysis of applications.

- hpcrun: a tool for profiling executions of unmodified application binaries using statistical sampling of hardware performance counters.
- csprof: a tool for collecting call path profiles of optimized application binaries.
- bloop: a tool for analyzing application binaries to recover program structure; namely, to identify where loops are present and what program source lines they contain.
- hpcprof: a tool for interpeting sample-based execution profiles and relating them back to program source lines.
- hpcview: a tool for correlating program structure information, multiple sample-based performance profiles, and program source code to produce a performance database.

© Bull 2009

First Use

Compile and Run

- Compile with -g
- run with hpcrun ./a.out with default events (PAPI_TOT_CYC)
- Result inside file a.out.PAPI_TOT_CYC.lynx0.25098.0x620a

Analysis 1/2

- Run hpcprof ./a.out a.out.PAPI_TOT_CYC.lynx0.25098.0x620a
- Result: ASCI or HTML

```
Columns correspond to the following events [event:period (events/sample)]
  PAPI_TOT_CYC:32767 - Total cycles (5162 samples)

Load Module Summary:
  99.8% /<path_to_bin>/cg.S
   0.2% /lib/ld-2.3.4.so
[...]
File Summary:
  98.7% <<path_to_bin>/cg.S>><path_to_src>/cg.f
   0.9% <<path_to_bin>/cg.S>><path_to_src>/randi8.f
[...]
```


© Bull 2009

Analysis 2/2

```
Function Summary:
  85.5% <</path_to_src>/cg.S>>conj_grad
  12.5% <<<path_to_bin/cg.S>>sparse
[...]

File <<path_to_bin/cg.S>><path_to_src>/cg.f with profile annotations.

   516         c---------------------------------------------------------------------
   517         c  q = A.p
   518         c  The partition submatrix-vector multiply: use workspace w
   519         c---------------------------------------------------------------------
   520         C
   521         C  NOTE: this version of the multiply is actually (slightly: maybe %5)
   522         C        faster on the sp2 on 16 nodes than is the unrolled-by-2 version
   523         C        below.   On the Cray t3d, the reverse is true, i.e., the
   524         C        unrolled-by-two version is some 10% faster.
   525         C        The unrolled-by-8 version below is significantly faster
   526         C        on the Cray t3d - overall speed of code is 1.5 times faster.
   527         C
   528   0.6%           do j=1,lastrow-firstrow+1
   529                     sum = 0.d0
   530  29.3%              do k=rowstr(j),rowstr(j+1)-1
   531  49.8%                 sum = sum + a(k)*p(colidx(k))
   532                     enddo
   533   1.1%              q(j) = sum
   534                 enddo
[...]
```


© Bull 2009

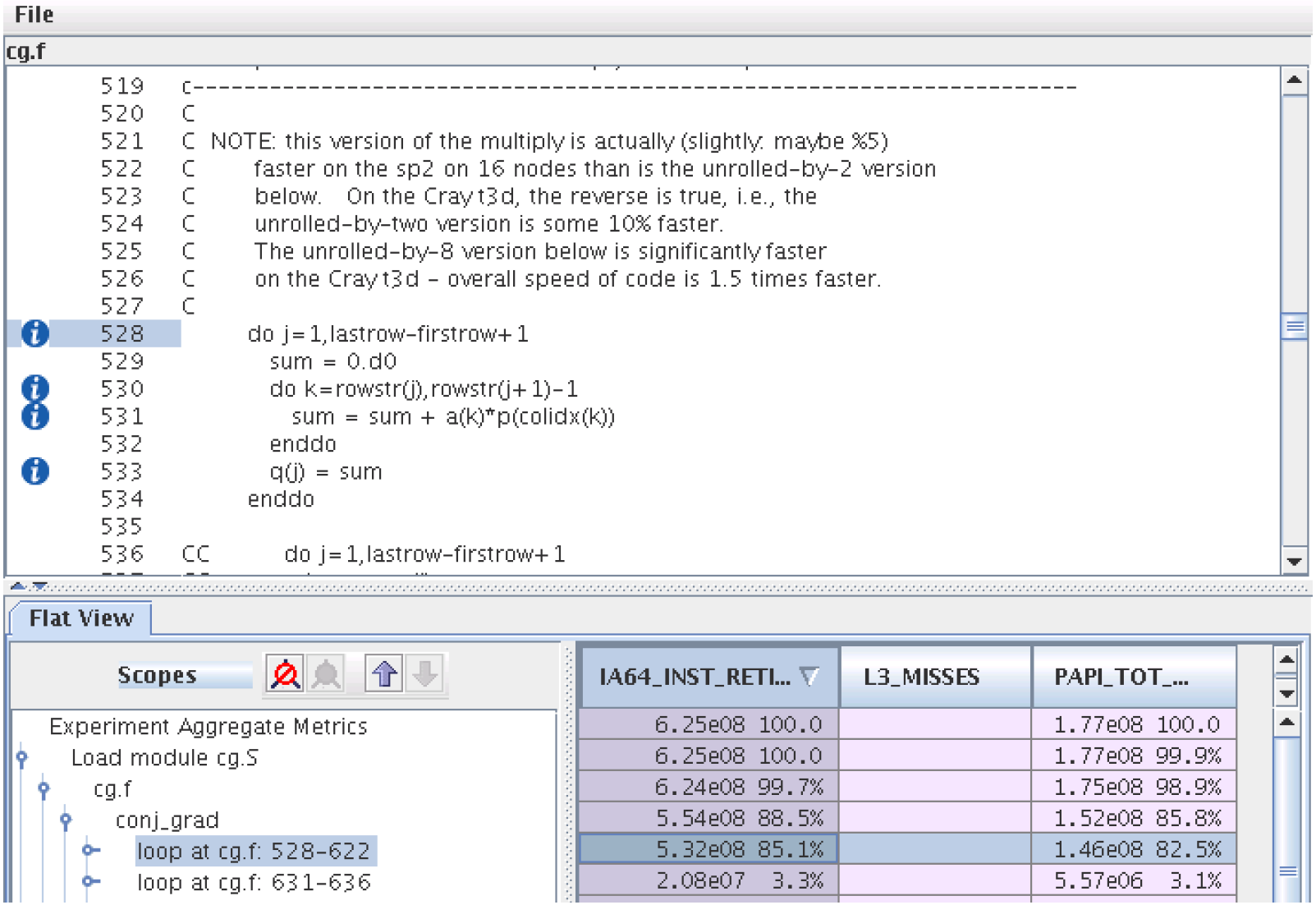
Use GUI

Run

- Analyze application binaries:
  `bloop ./a.out > a.out.psxml`
- Run: `hpcrun -e events1 -e events2 -e events3 ./a.out>profile`
- Run: `hpcquick -I<SRC_PATH> -S a.out.psxml -P profile -n`
- Run: `hpcview -o experiment-db hpcquick.xml`
- Run: `hpcviewer`

© Bull 2009

Hpcviewer



© Bull 2009

Key Events in action

- Total cycles = Cycles issuing μOps + Cycles not issuing μOps

$$CPU_CLK_UNHALTED.CORE \simeq RS_UOPS_DISPATCHED.CYCLES_ANY + RS_UOPS_DISPATCHED.CYCLES_NONE$$

- Non-productive Cycles (cycles exe but not retired)

$$RS_UOPS_DISPATCHED - (UOPS_RETIRED.ANY + UOPS_RETIRED.ANY.FUSED)$$

- Clocks per Instruction (CPI)

$$\frac{INST_RETIRED.ANY}{CPU_CLK_UNHALTED.CORE}$$

- Bus Utilization

$$\frac{BUS_TRANS_ANY.ALL_AGENTS \times 2}{CPU_CLK_UNHALTED.BUS} \times 100$$

© Bull 2009

Key Events in action

- FSB Saturation

$$\frac{64 \times BUS_TRANS_ANY.ALL_AGENTS \times freq}{CPU_CLK_UNHALTED.CORE}$$

- L2 cache Miss Rate and demand miss rate

$$\frac{L2_LINES_IN.SELF.ANY}{INST_RETIRED.ANY}$$

$$\frac{L2_LINES_IN_SELF.DEMAND}{INST_RETIRED.ANY}$$

- Resource Stall Ratio

$$\frac{RESOURCE_STALLS.ANY}{CPU_CLCK_UNHALTED.CORE}$$

© Bull 2009

- Introduction
- About Intel Compilers
- Help to port
- Optimization
- Bug Report
- MPIBull2
- Profiling inside the node
- Profiling : MPI
  Intel Trace Analyzer
  MPI Analyser

© Bull 2009

Communications: advanced Level

Intel Trace Tools : Collector ans Analyser

- Timeline Views and Parallelism Display
- Communication Statistics
- Execution Statistics
- Profiling Library
- MPI Bull support Intel Trace Tools



© Bull 2009

How to use it ?

Level 1

- Environment: source itacvars.sh or use a module file
- Link with -L$VT_LIB_DIR -lVT -ldwarf -lelf -lvtunwind -lnsl -lm -ldl -lpthread or $VT_ADD_LIBS
- Run your application
- Run traceanalyzer

© Bull 2009

Level 2 with Class

- At the beginning

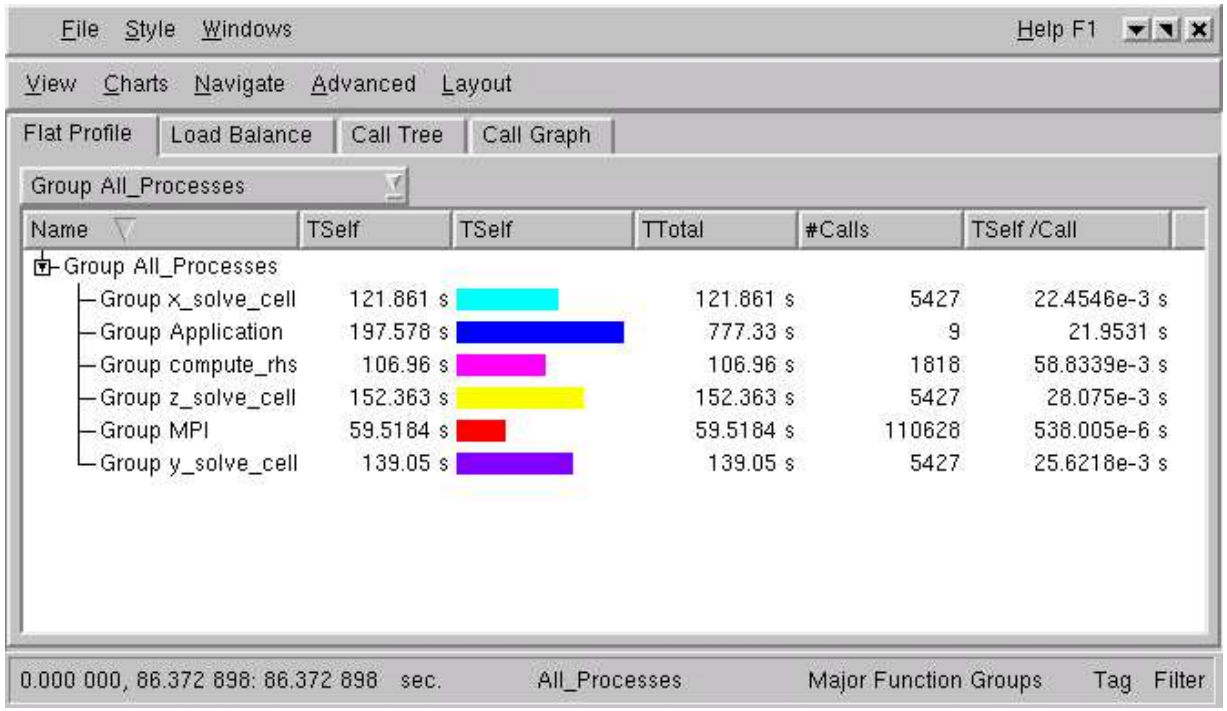
```
include "VT.inc"
integer function1,VT_ierr,class1
call VTCLASSDEF("GODFINE1",class1,VT_ierr)
call VTFUNCDEF("GODFINE1",class1,function1,VT_ierr)
call VTENTER(function1,VT_NOSCL,VT_ierr)
```

- At the end

```
call VTLEAVE(function1,VT_ierr)
```

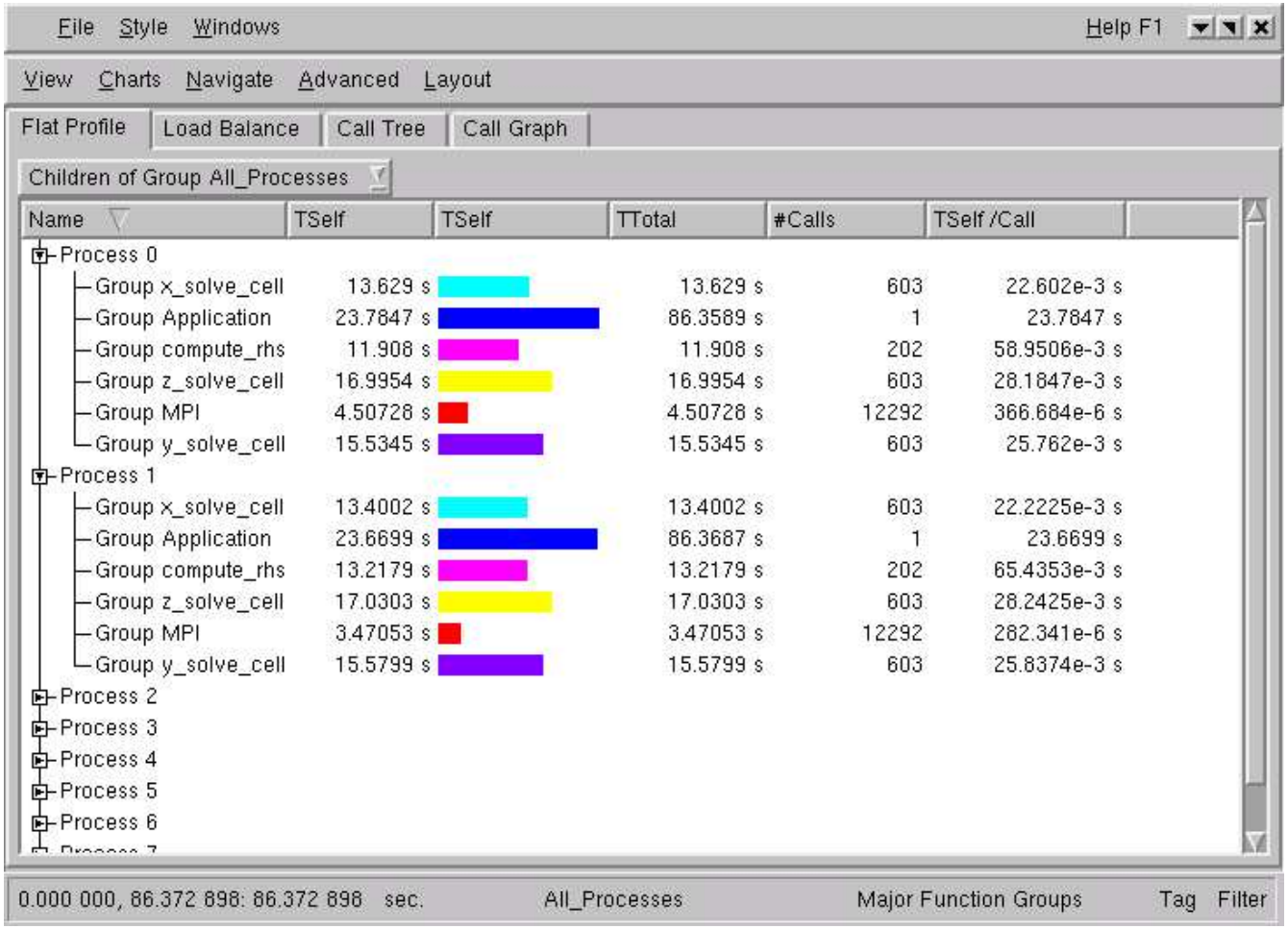

© Bull 2009

Example BT from NAS



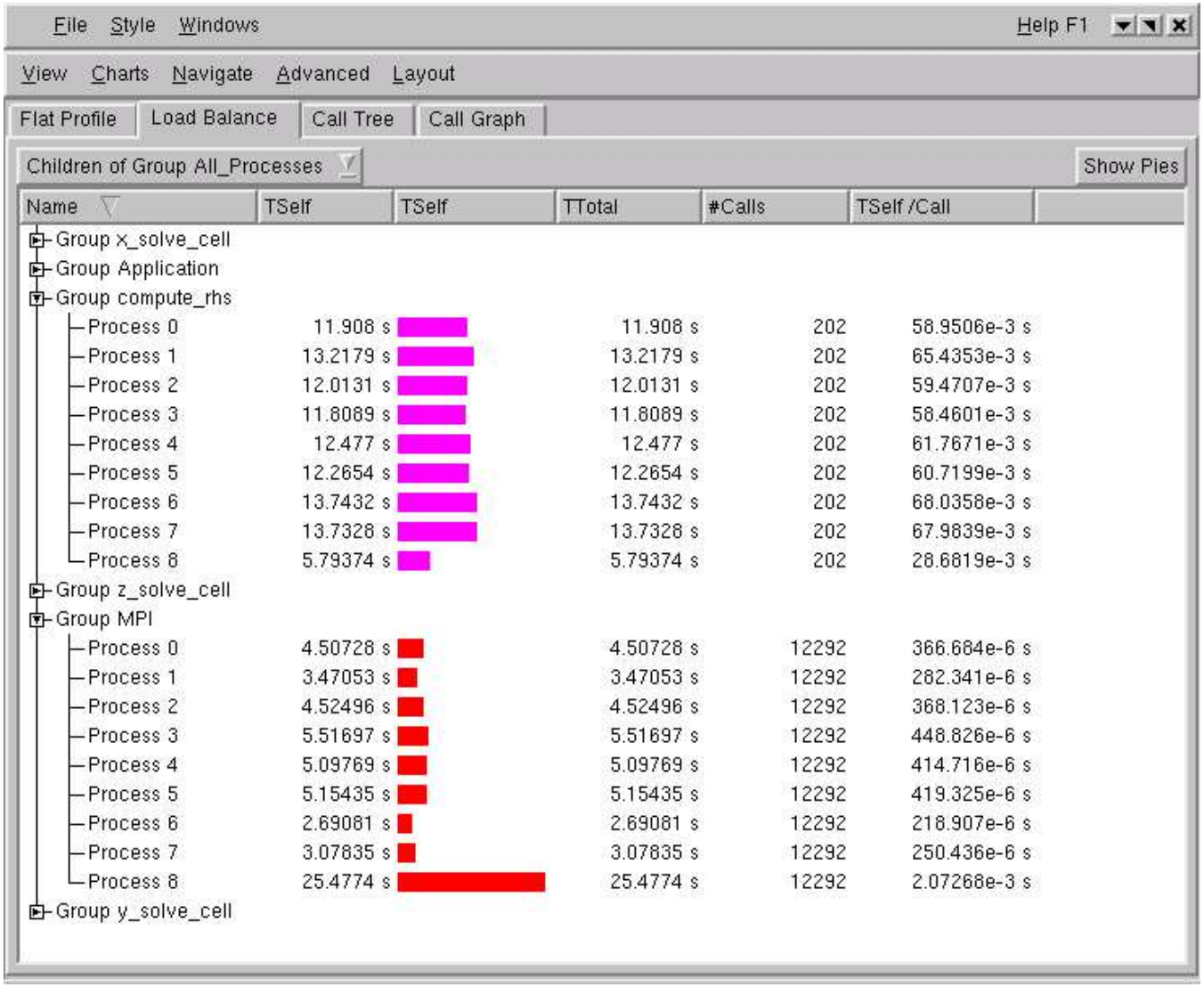
© Bull 2009

Example



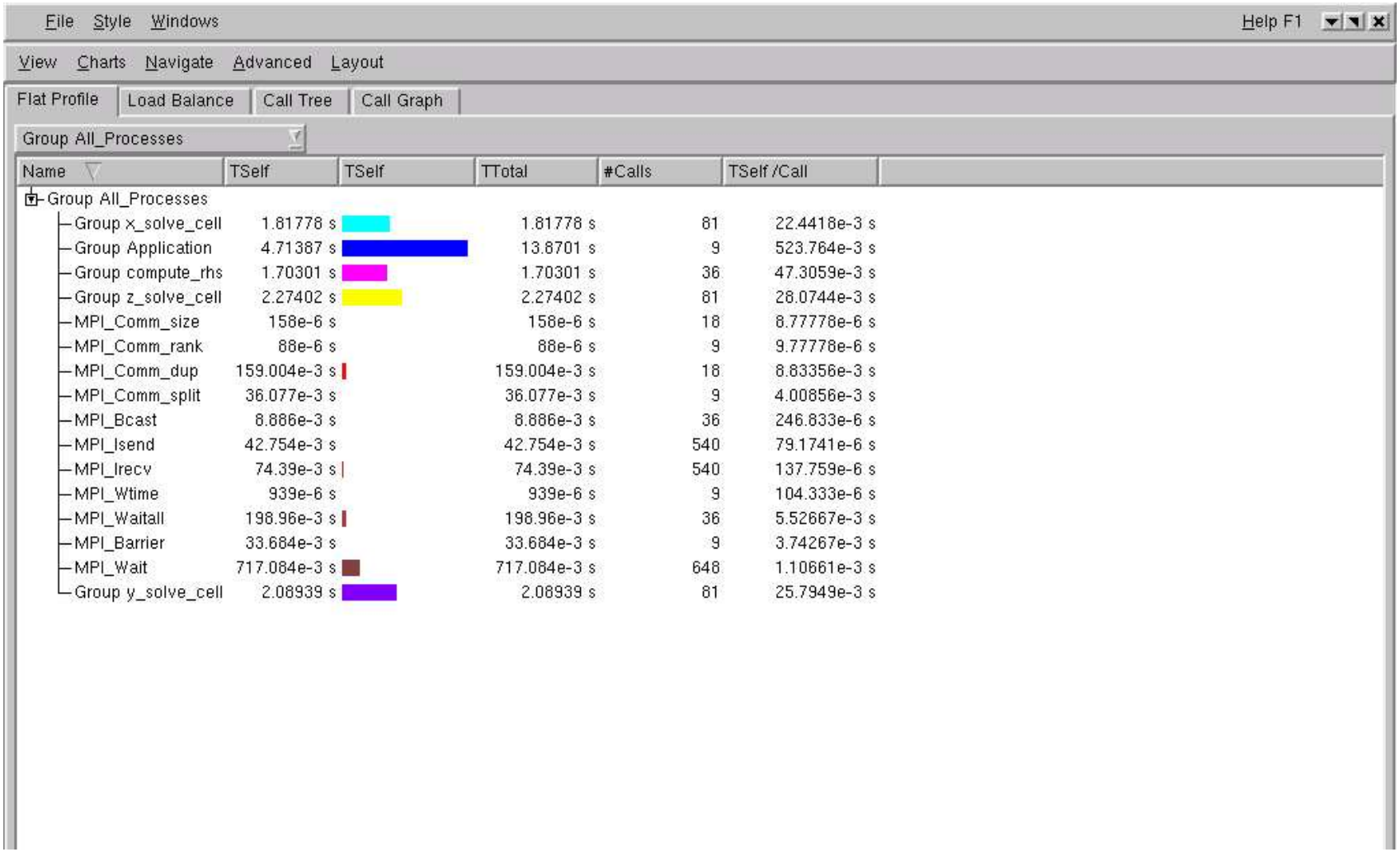
© Bull 2009

Example: Load balance



© Bull 2009

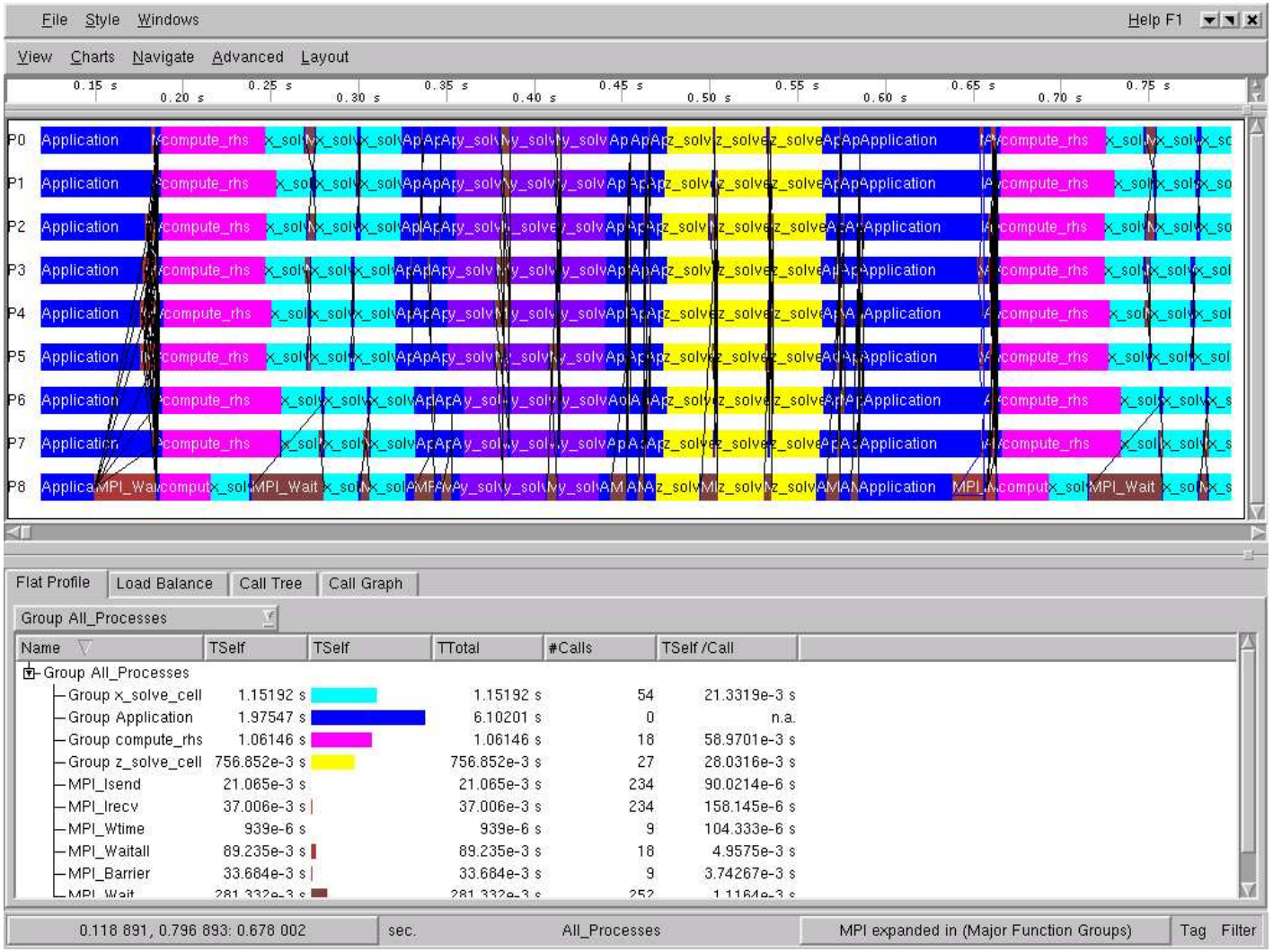
Example: Ungroup MPI



Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
Group All_Processes					
Group x_solve_cell	1.81778 s		1.81778 s	81	22.4418e-3 s
Group Application	4.71387 s		13.8701 s	9	523.764e-3 s
Group compute_rhs	1.70301 s		1.70301 s	36	47.3059e-3 s
Group z_solve_cell	2.27402 s		2.27402 s	81	28.0744e-3 s
MPI_Comm_size	158e-6 s		158e-6 s	18	8.77778e-6 s
MPI_Comm_rank	88e-6 s		88e-6 s	9	9.77778e-6 s
MPI_Comm_dup	159.004e-3 s		159.004e-3 s	18	8.83356e-3 s
MPI_Comm_split	36.077e-3 s		36.077e-3 s	9	4.00856e-3 s
MPI_Bcast	8.886e-3 s		8.886e-3 s	36	246.833e-6 s
MPI_Isend	42.754e-3 s		42.754e-3 s	540	79.1741e-6 s
MPI_Irecv	74.39e-3 s		74.39e-3 s	540	137.759e-6 s
MPI_Wtime	939e-6 s		939e-6 s	9	104.333e-6 s
MPI_Waitall	198.96e-3 s		198.96e-3 s	36	5.52667e-3 s
MPI_Barrier	33.684e-3 s		33.684e-3 s	9	3.74267e-3 s
MPI_Wait	717.084e-3 s		717.084e-3 s	648	1.10661e-3 s
Group y_solve_cell	2.08939 s		2.08939 s	81	25.7949e-3 s

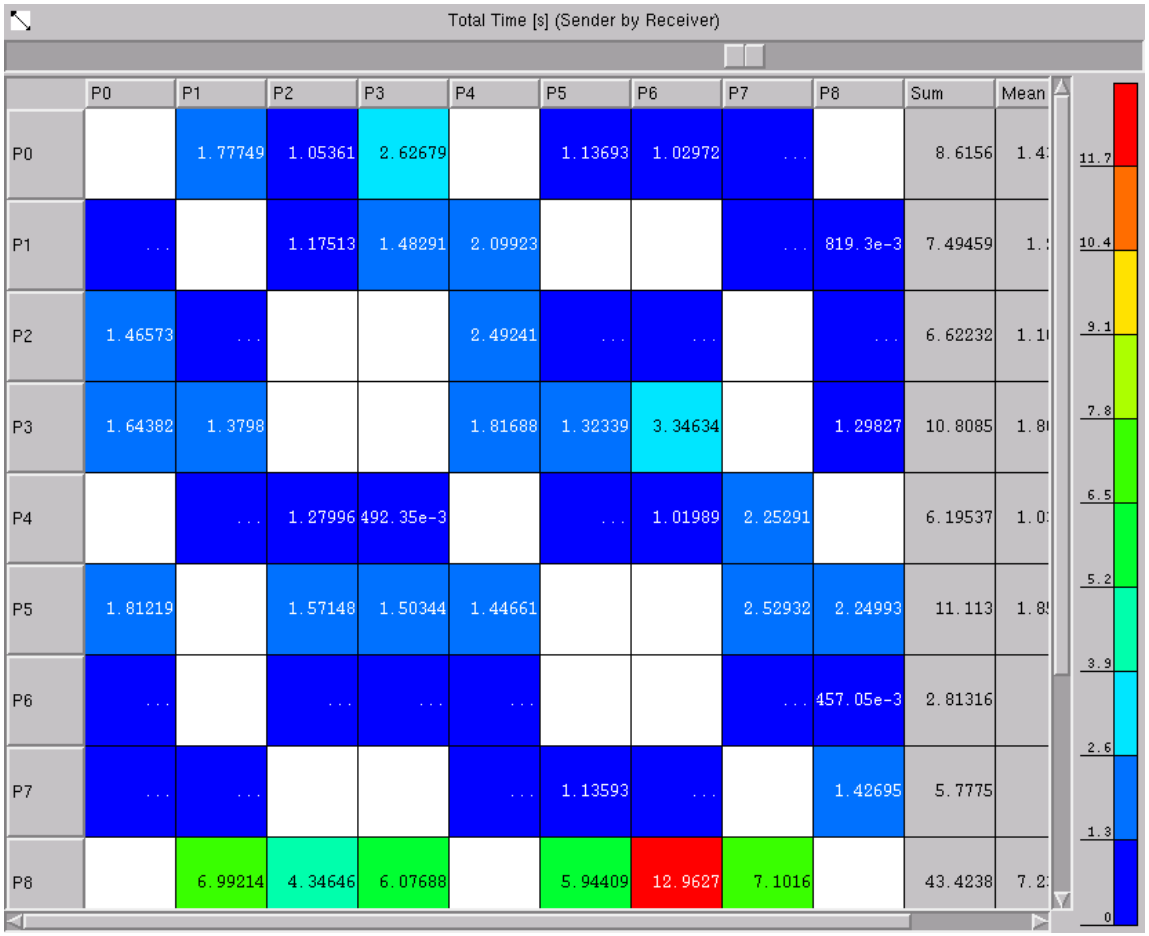
© Bull 2009

Example



© Bull 2009

Example



© Bull 2009

With next MPI Bull version

MPIAnalyser

MPIAnalyser is a profiling tool dedicated to MPI applications.

- Light: it uses few resources and so does not slow down the application.
- Easy to run: it is used to characterize the MPI communications in a program. Communication matrices are constructed with it. Profilecomm is a "post-mortem" tool.

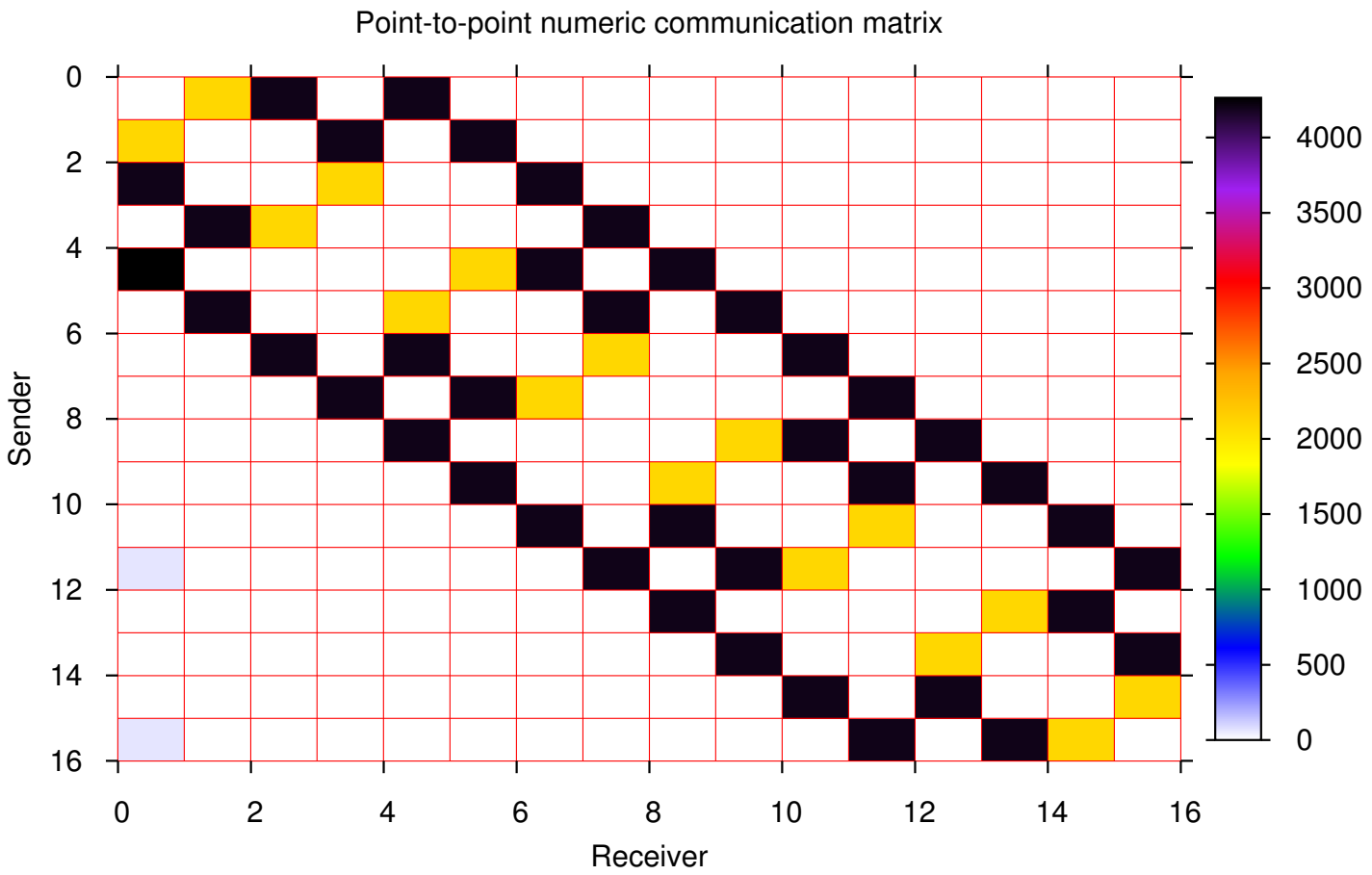
© Bull 2009

How to use it ?

- Link with -lmpianalyser
- Use environment
  - MPIANALYSER_PROFILECOMM=1
  - MPIANALYSER_GLOBAL=1
- Run your application
- Use tool: readpfc

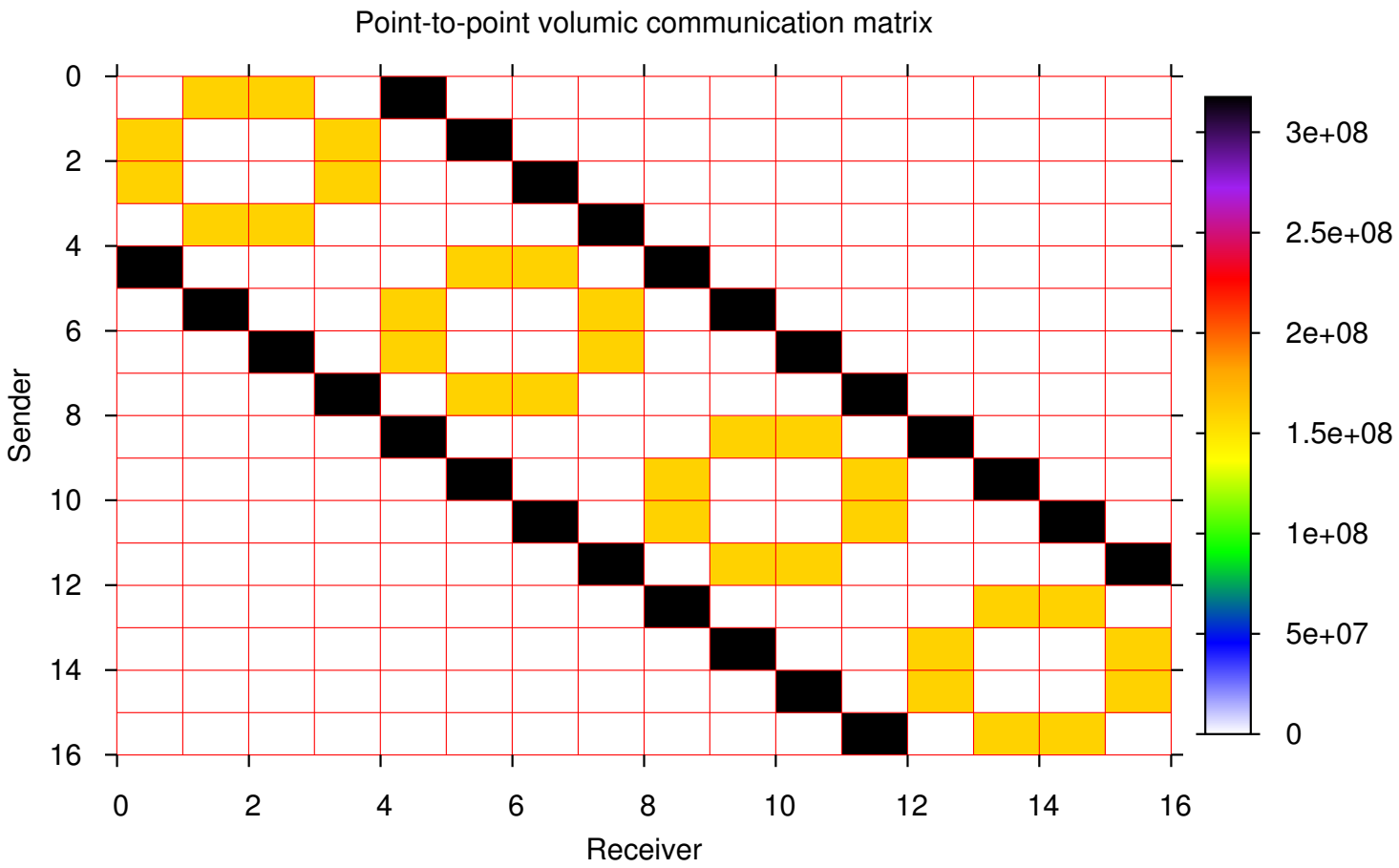
© Bull 2009

Point to Point numeric command matrix



© Bull 2009

Point to Point Volumic communication matrix



© Bull 2009



Architect of an Open World™

© Bull 2009

- (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

© Bull 2009