High Performance Profiling Tools

Georges Markomanolis
Master internship
Graal Team

September 7, 2009

Supervisors: Frédéric Desprez and Frédéric Suter

Table of contents

Introduction

- Many profiling tools in order to profile/trace a parallel application
- Communication bottlenecks
- TAU and Scalasca
- In the next sections an application will be profiled with both programs and a comparison of overhead will take place at the end.

Performance Application Programming Interface (PAPI)

PAPI aims to provide the tool designer and application engineer with a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors.

PAPI Events

- PAPI supports 103 events.
- The events are hardware counters on the cpu.
- Every cpu has 2-8 counters.

NAS Parallel Benchmarks

The NAS Parallel Benchmarks (NPB) are a set of programs in order to evaluate the performance of parallel computers. The benchmarks are derived from computational fluid dynamics and they consist of five kernels and three applications. NAS Parallel Benchmarks are executed on Grid'5000.

NAS Parallel Benchmarks

- EP, this code implements the random-number generator. The code is "embarrassingly" parallel in that no communication is required for the generation of the random numbers itself. There is no special requirement on the number of processors used for running the benchmark.
- FT, this code implements the time integration of a three-dimensional partial differential equation using the Fast Fourier Transform.
- Data Traffic benchmark DT is written in C. DT benchmark takes one argument: BH, WH, or SH. This argument specifies the communication graph Black Hole, White Hole, or SHuffle respectively.
- LU, a regular-sparse, block (5 x 5) lower and upper triangular system solution. This problem represents the computations associated with the implicit operator of a newer class of implicit CFD algorithms.

NAS's Classes

There are 7 classes at the NAS: W,S,A,B,C,D,E. Each one represent the size of the problem (input data).

- The W,S are very small classes.
- The A,B classes are used for a small number of processors.
- The C,D classes are used for executing big benchmarks.
- The class E is for huge benchmarks

TAU

- Tuning and Analysis Utilities
- Profiling and tracing toolkit for performance analysis of parallel programs written in C, C++, Fortran, Java and Python.
- Support for multiple parallel programming paradigms: MPI, Multi-threading, Hybrid.
- Access to hardware counters.
- Automatically instruments your code.
- Implemented by University of Oregon

Using TAU

- Declaration of environment variables such as $TAU_MAKEFILE, $TAU_OPTIONS.
- Selective instrumentation by inserting TAU macros or automatically.
- Program Database Toolkit (PDT).
- For automatic instrumentation replace the compiler with TAU compiler script
- Many flags for configuration

Using TAU (II)

TAU Shell Scripts

GNU Compilers	TAU shell scripts
mpicc	tau_cc.sh
mpicxx	tau_cxx.sh
mpif77 mpif90	tau_f90.sh

TAU Instrumentation

- Support for standard programs events
  - Routines, classes and templates
  - Statement-level blocks
  - Begin/End events
- Support for user-defined events
  - Begin/End events specified by user
  - Atomic events (e.g., size of memory)
- Static events and dynamic events

MPI Wrapper Interposition Library

- Standard MPI Profiling Interface
- TAU instrumented MPI library
  - Interpose between MPI and TAU
  - During program link, -lmpi replaced by -lTauMpi -lpmpi -lmpi
  - No change to the source code.

Selective Instrumentation File

- Specify a list of events or files to exclude or include

```
BEGIN_EXCLUDE_LIST
Verify
Tak#
common.f
END_EXCLUDE_LIST
BEGIN_INCLUDE_LIST
comp#
ff*.f
END_INCLUDE_LIST
```

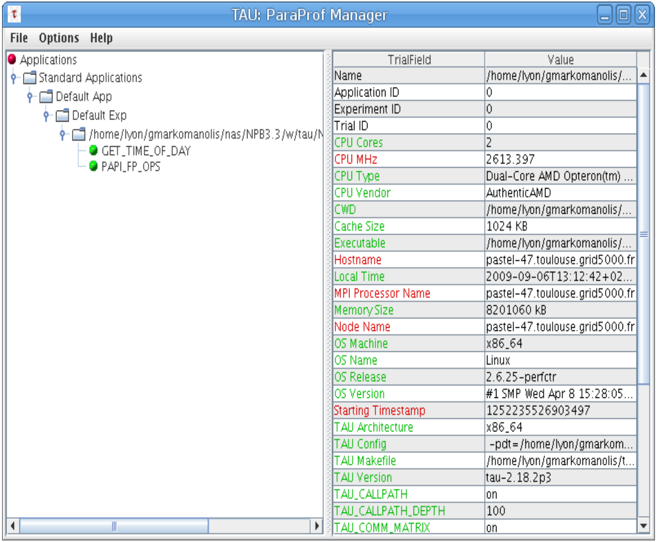
- The # is wildcard for in a routine name and the * is wildcard character

Selective Instrumentation File (II)

- Specify a list of user instrumentation commands

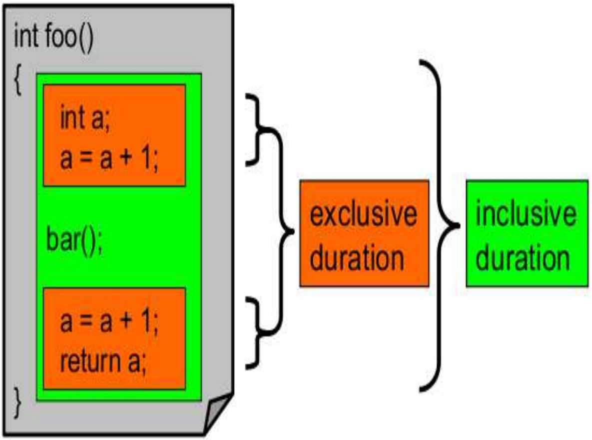
```
BEGIN_INSTRUMENT_SECTION
dynamic phase name"test1" file="ft.f" \
line=148 to line=173
loops file="ft.f" routine="#"
END_INSTRUMENT_SECTION
```

Demo - Paraprof manager for FT benchmark, class C



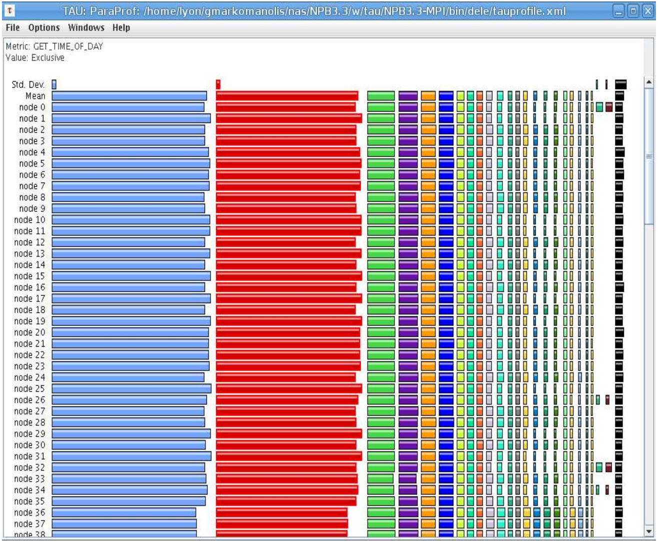
TrialField	Value
Name	/home/lyon/gmarkomanolis/...
Application ID	0
Experiment ID	0
Trial ID	0
CPU Cores	2
CPU MHz	2613.397
CPU Type	Dual-Core AMD Opteron(tm) ...
CPU Vendor	AuthenticAMD
CWD	/home/lyon/gmarkomanolis/...
Cache Size	1024 KB
Executable	/home/lyon/gmarkomanolis/...
Hostname	pastel-47.toulouse.grid5000.fr
Local Time	2009-09-06T13:12:42+02...
MPI Processor Name	pastel-47.toulouse.grid5000.fr
Memory Size	8201060 kB
Node Name	pastel-47.toulouse.grid5000.fr
OS Machine	x86_64
OS Name	Linux
OS Release	2.6.25-perfctr
OS Version	#1 SMP Wed Apr 8 15:28:05...
Starting Timestamp	1252235526903497
TAU Architecture	x86_64
TAU Config	-pdt=/home/lyon/gmarkom...
TAU Makefile	/home/lyon/gmarkomanolis/t...
TAU Version	tau-2.18.2p3
TAU_CALLPATH	on
TAU_CALLPATH_DEPTH	100
TAU_COMM_MATRIX	on

Demo - Explanation of exclusive and inclusive time



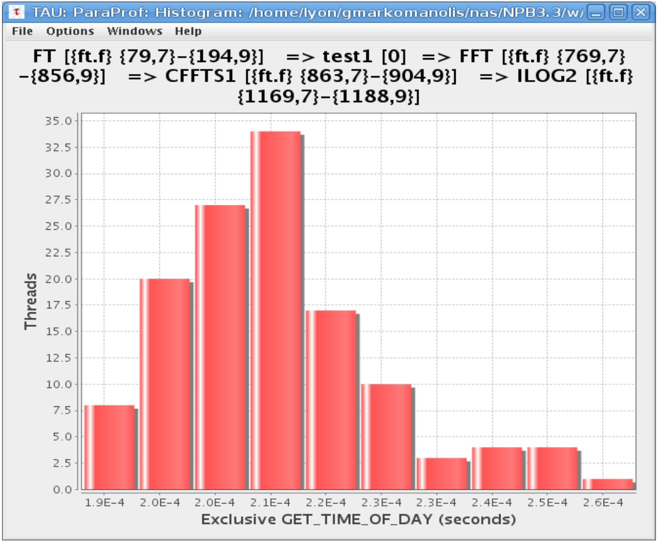
Demo - Exclusive time

Demo - Exclusive time



Demo - Exclusive time for an event (MPI_Alltoall)

Demo - Histogram for ILOG2()



Demo - Thread 0 Statistics

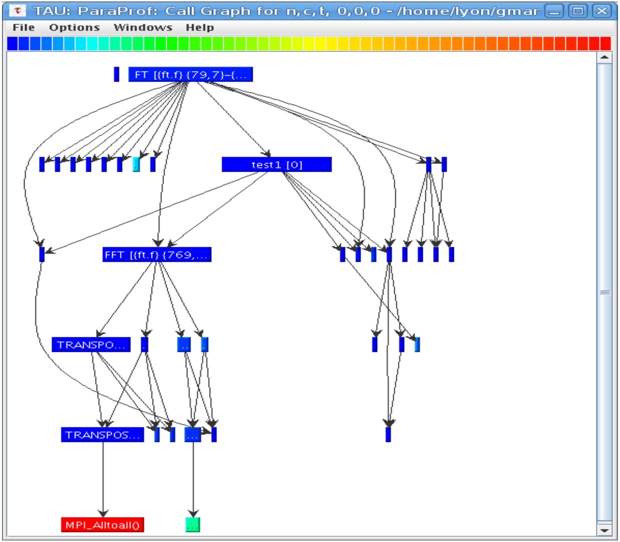
TAU: ParaProf: Thread Statistics: n,c,t, 0,0,0 - /home/lyon/gmarkomanolis/nas/NPB3.3/w/tau/NPB3.3-MPI/bin/(

File Options Windows Help

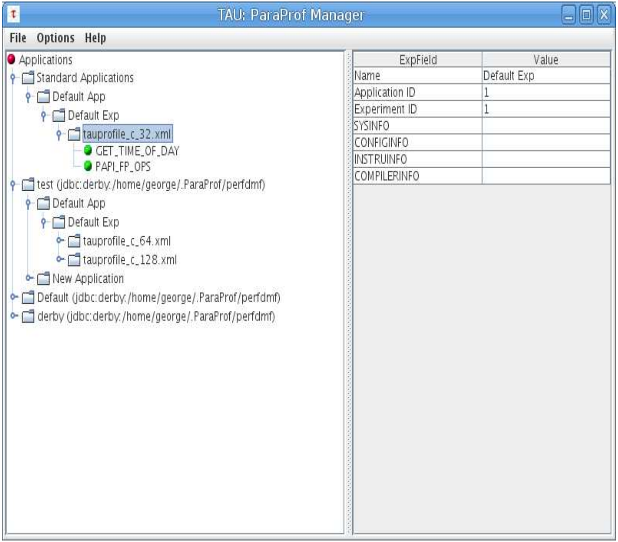
GET_TIME_OF_DAY

Name △	Exclusive GET_TI...	Inclusive GET_TI...	Exclusive PAPI_FP...	Inclusive PAPI_FP...	Calls	Child ...
FT [{ft.f} {79,7}-{194,9}]	0.317	35.374	6,310	7,099,274,870	1	44
COMPUTE_INDEXMAP [{ft.f} {647,7}-{722,9}]	0.071	0.071	68,326,462	68,326,462	1	0
COMPUTE_INITIAL_CONDITIONS [{ft.f} {233,7}-{280,9	0	0.026	589	2,115,786	1	10
FFT [{ft.f} {769,7}-{856,9}]	0	1.294	248	311,728,188	1	4
FFT_INIT [{ft.f} {1004,7}-{1043,9}]	0	0	21,337	21,354	1	1
MPI_Barrier()	0.033	0.033	19	19	1	0
MPI_Finalize()	0.055	0.055	4,644	4,644	1	0
MPI_Init()	2.287	2.287	678	678	1	0
PRINT_RESULTS [{print_results.f} {2,7}-{114,12}]	0	0	48	48	1	0
SETUP [{ft.f} {343,7}-{641,9}]	0.001	0.002	385	631	1	6
MPI_Bcast()	0	0	172	172	2	0
MPI_Comm_rank()	0	0	20	20	1	0
MPI_Comm_size()	0	0	20	20	1	0
MPI_Comm_split()	0.001	0.001	34	34	2	0
TIMER_CLEAR [{timers.f} {4,7}-{17,9}]	0	0	569	569	30	0
TIMER_READ [{timers.f} {65,7}-{77,9}]	0	0	19	19	1	0
TIMER_START [{timers.f} {23,7}-{37,9}]	0	0	21	21	1	0
TIMER_STOP [{timers.f} {43,7}-{59,9}]	0	0	27	27	1	0
VERIFY [{ft.f} {1750,7}-{1991,9}]	0	0	1,949	1,968	1	1
MPI_Comm_size()	0	0	19	19	1	0
test1 [0]	0.001	31.289	3,655	6,717,068,146	1	64
CHECKSUM [{ft.f} {1685,7}-{1726,9}]	0.003	1.062	2,707	4,880	20	20
COMPUTE_INDEXMAP [{ft.f} {647,7}-{722,9}]	0.067	0.067	68,238,983	68,238,983	1	0
COMPUTE_INITIAL_CONDITIONS [{ft.f} {233,7}-{28	0	0.014	593	2,099,507	1	10
EVOLVE [{ft.f} {199,7}-{227,9}]	0.383	0.383	84,049,973	84,049,973	20	0
FFT [{ft.f} {769,7}-{856,9}]	0.001	29.762	5,222	6,562,649,797	21	84
FFT_INIT [{ft.f} {1004,7}-{1043,9}]	0	0	21,332	21,351	1	1

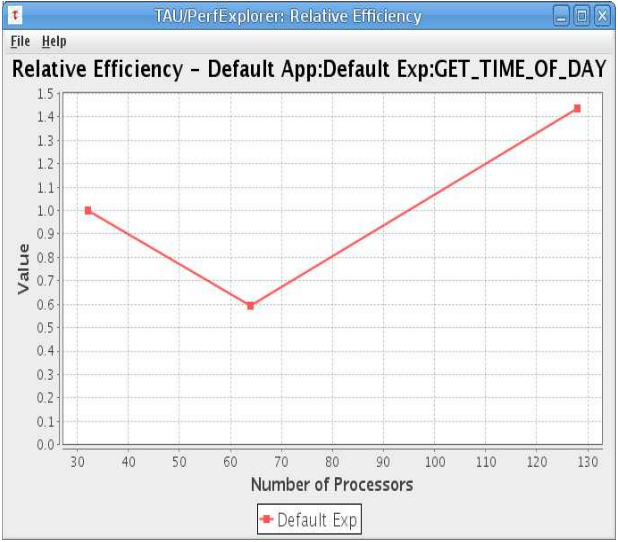
Demo - Call Graph



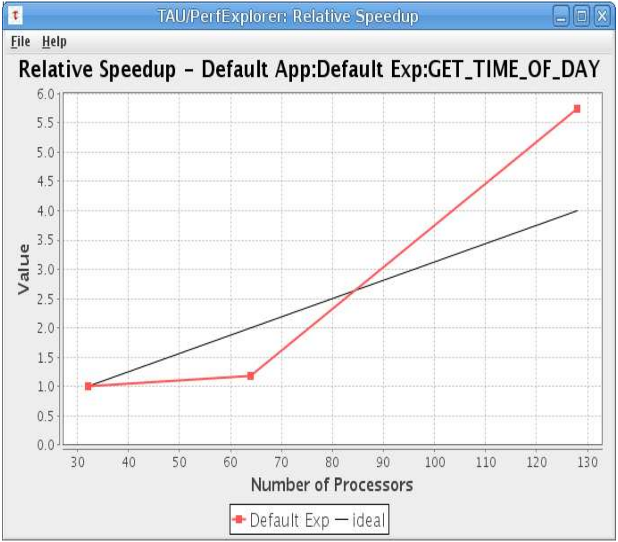
Demo - PerfDMF



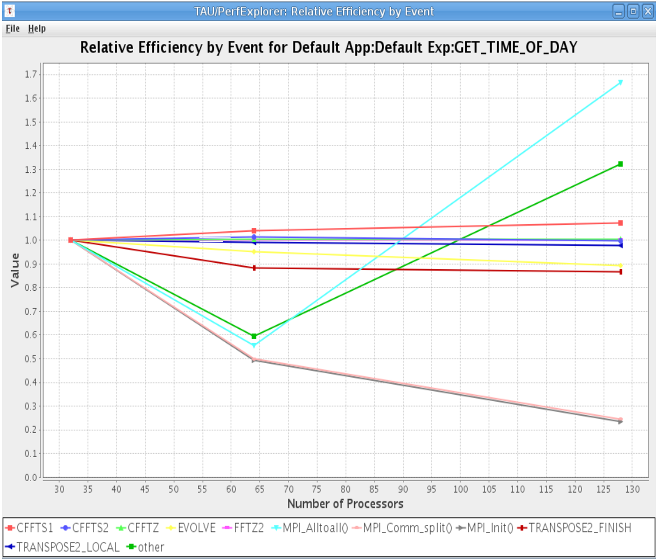
Demo - PerfExplorer, Efficiency



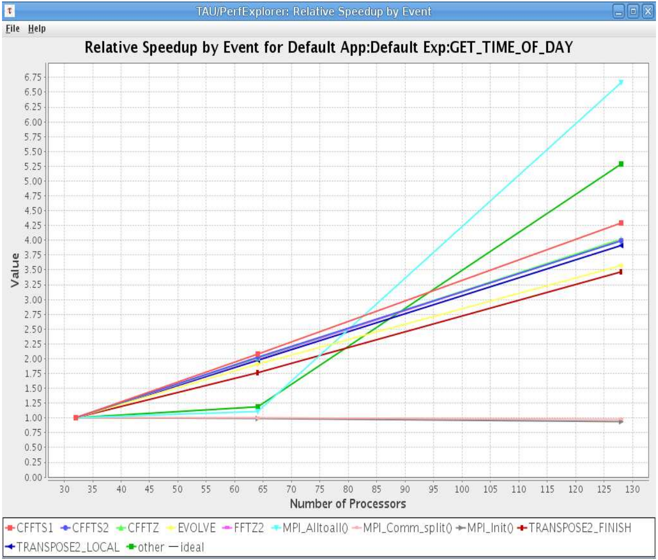
Demo - PerfExplorer, Speedup



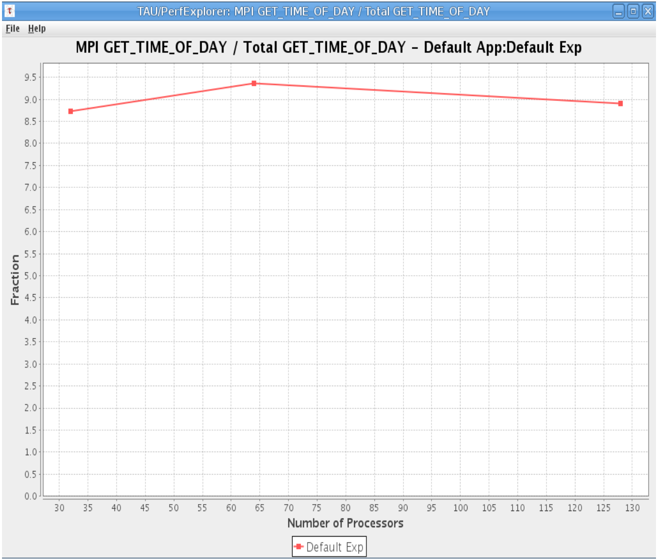
Demo - PerfExplorer, Efficiency by event



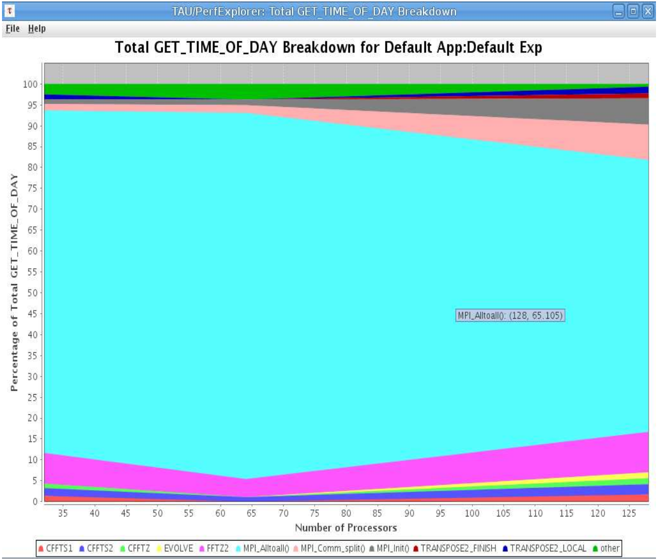
Demo - PerfExplorer, Speedup by event



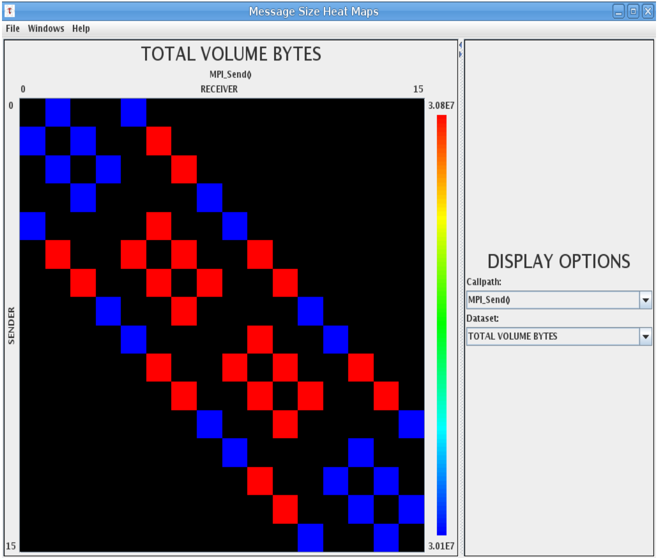
Demo - MPI Time / Total time



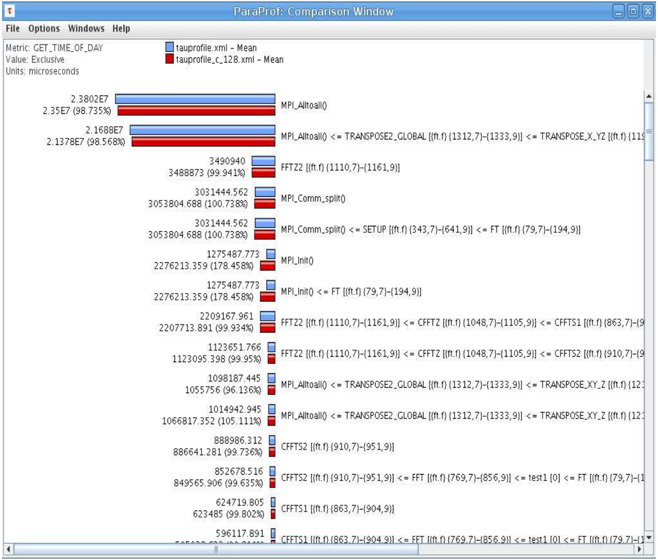
Demo - The Runtime Breakdown chart



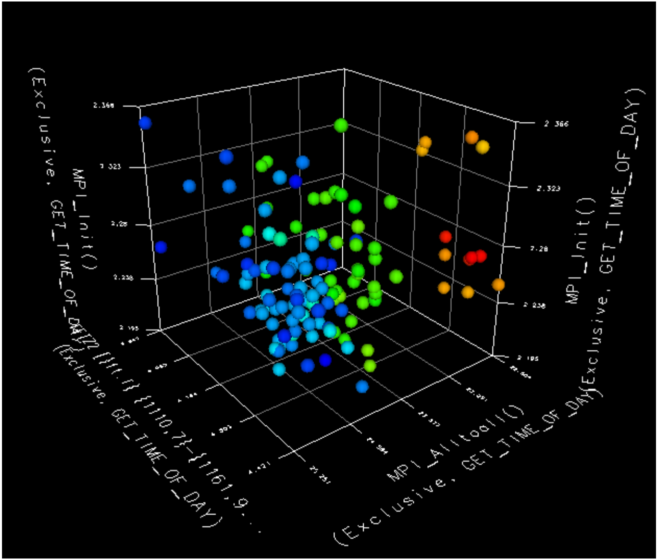
Demo - Communication Matrix



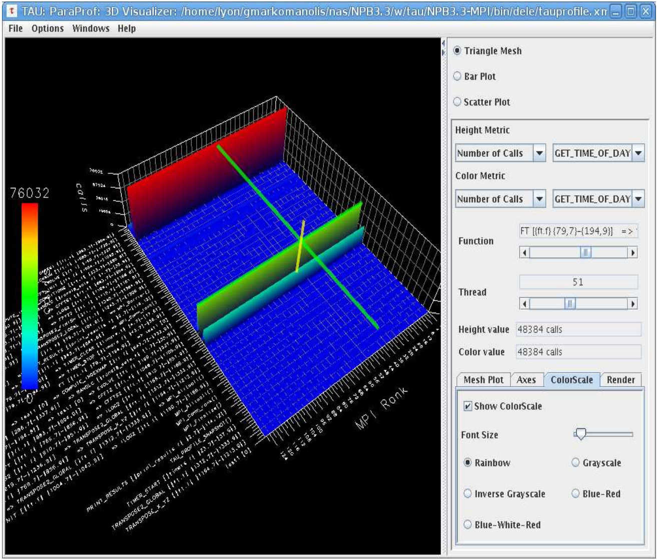
Demo - Comparison of two executions



Demo - Visualization I



Demo - Visualization II

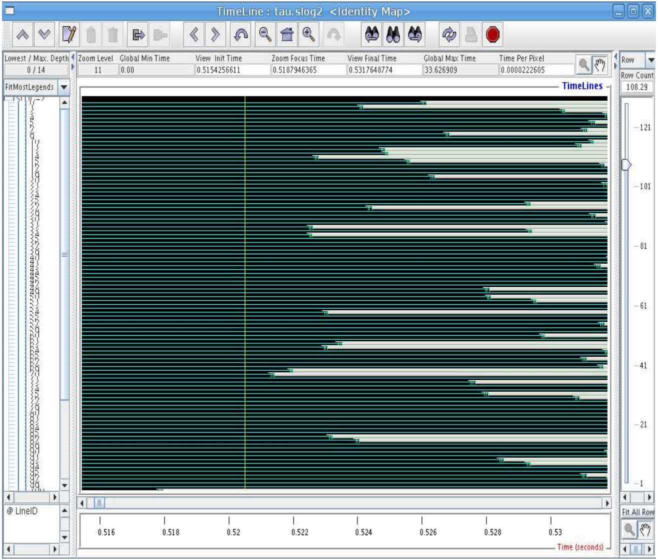


Demo - Tracing

- One way to visualize the trace output from TAU is the Jumpshot (integrated with TAU).
- Supported file formats: OTF, Vampir Trace Format, EPILOG, Slog2
- For creating slog2 it is needed to merge all the trace files.

Demo - Visualizing trace files with Jumpshot

Demo - Visualizing trace files with Jumpshot



Pros and Cons

- Pros
  - Visualizing MPI calls is great help
  - User can identify message sizes, time taken
  - Automatic selective instrumentation
  - Communication Matrix
  - Tools for visualizing many useful data, save experiments to databases
  - Memory tracking
  - TAUCuda is on the way
- Cons
  - No automatic recognition of bottlenecks
  - Large trace files for visualising.
  - Slow tracefile reading
  - Bad structure of output file format

Scalasca

- Scalable Performance Analysis of Large-Scale Applications
- Profiling and tracing toolkit for performance analysis of parallel programs written in C, C++, Fortran.
- Profile analysis.
- Time-line analysis (trace)
- Automatic identification of bottlenecks
- Pattern analysis
- Automatically instruments your code.
- Implemented by Julich Supercomputing Centre

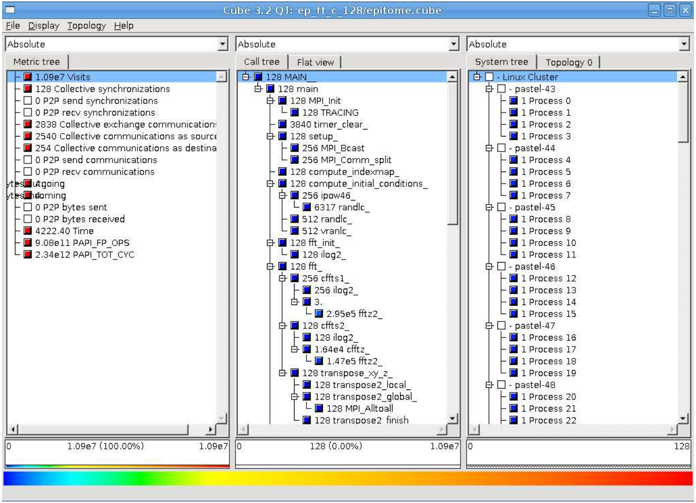
Using Scalasca

- Just add at your makefile a variable $PREP before mpicc and declare that PREP="scalasca -instrument"
- For executing your application with profiling, execute scalasca -analyze mpirun ...
- For executing your application with tracing, execute scalasca -analyze -t mpirun ...
- For analyzing profiling data, execute scalasca -examine name of folder with results of the above command.
- For analyzing tracing data, execute mpirun ... scout name of folder with results of the second command and after execute scalasca -examine name of folder.

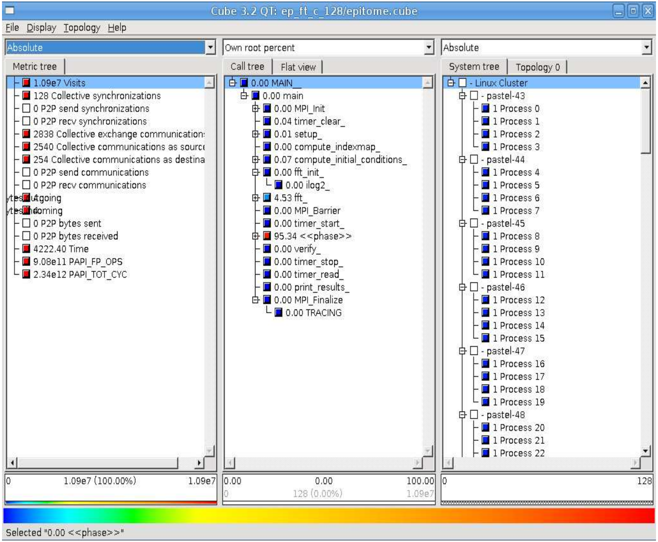
Selective Instrumentation File

- Scalasca does not support selective instrumentation as TAU, it is possible to declare only the functions that should be excluded.
- The good news is that it will be available soon and it will be implemented like TAU, using PDT for automatic instrumentation.

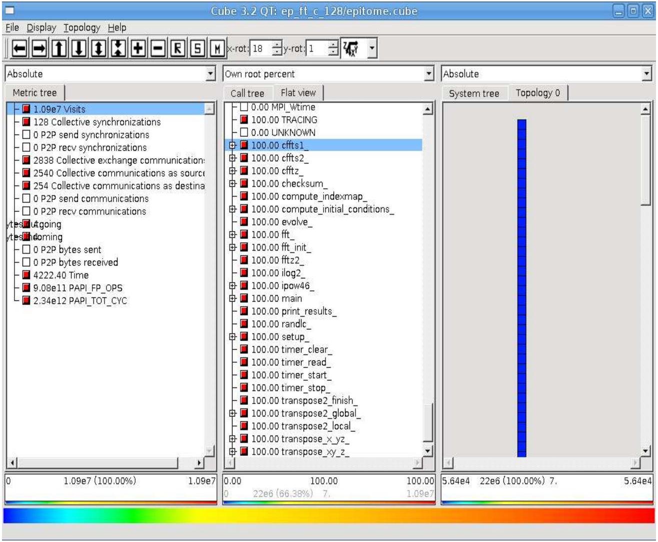
Demo - Parallel program analysis report exploration (Cube3) for FT benchmark, class C



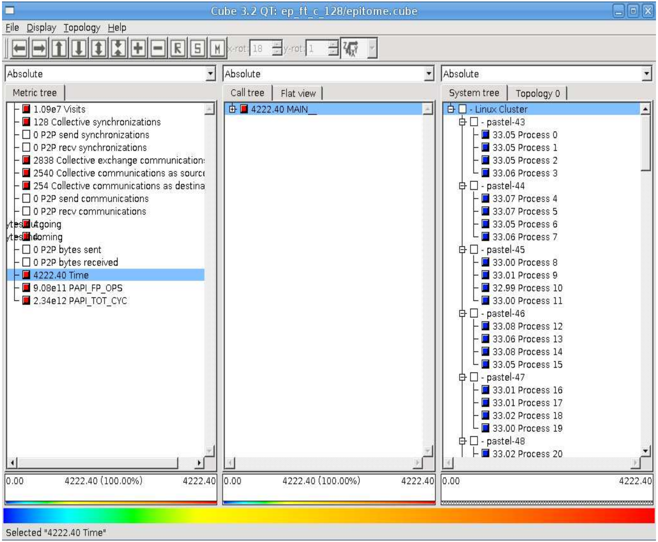
Demo - Selective instrumentation (phase)



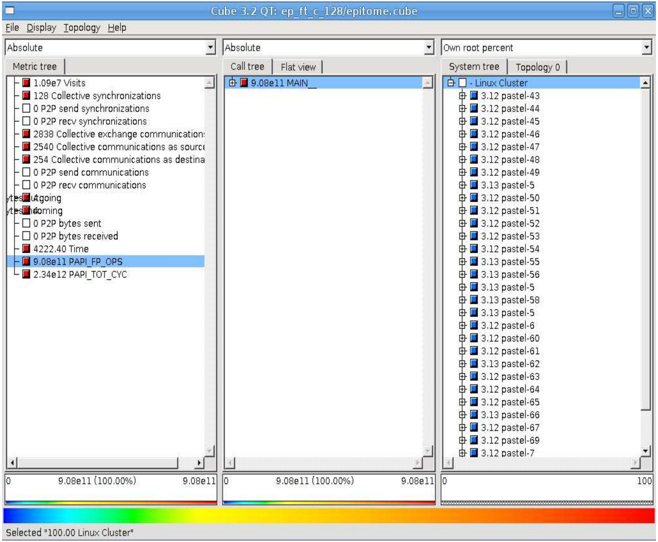
Demo - Topology



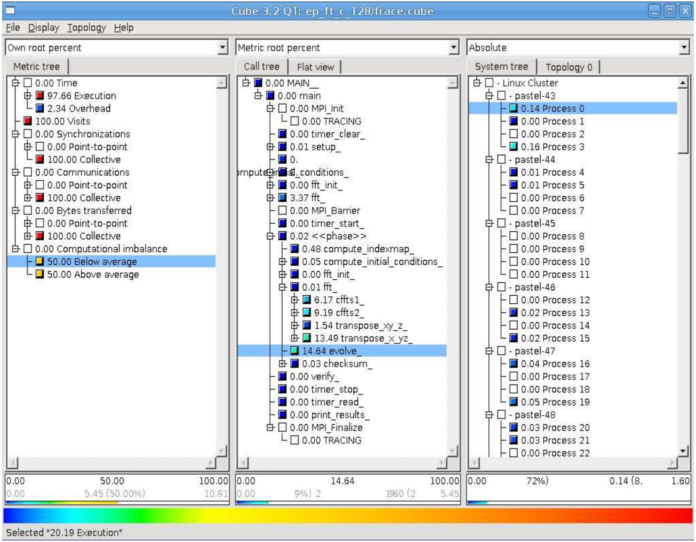
Demo - Load balancing I



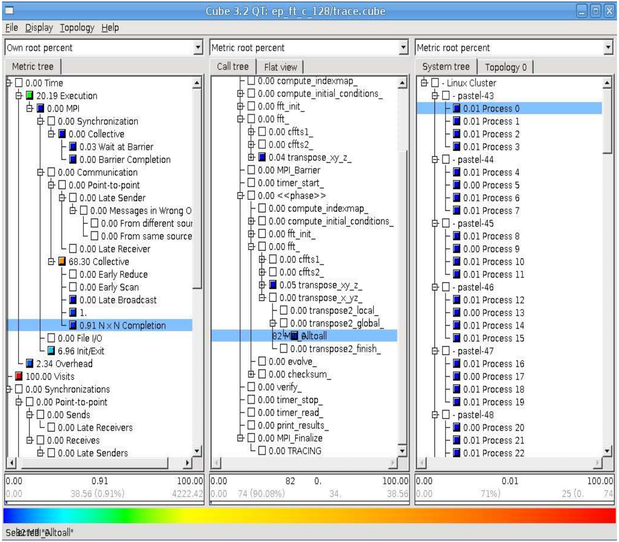
Demo - Load balancing II



Demo - Load balancing III



Demo - Late communication I

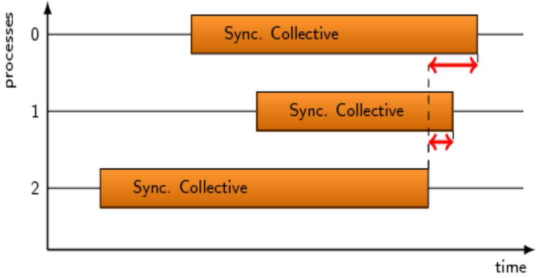


Demo - Late communication II

N x N Completion Time

Description:

This pattern refers to the time spent in MPI n-to-n collectives after the first process has left the operation.



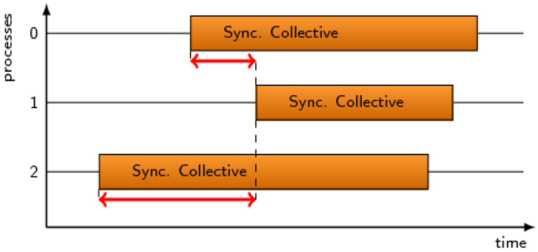
Note that the time reported by this pattern is not necessarily completely waiting time since some processes could -- at least theoretically -- still communicate with each other while others have already finished communicating and exited the operation.

Demo - Late communication III

Wait at N x N Time

Description:

Collective communication operations that send data from all processes to all processes (i.e., n-to-n) exhibit an inherent synchronization among all participants, that is, no process can finish the operation until the last process has started it. This pattern covers the time spent in n-to-n operations until all processes have reached it. It applies to the MPI calls MPI_Reduce_scatter(), MPI_Allgather(), MPI_Allgatherv(), MPI_Allreduce(), MPI_Alltoall(), MPI_Alltoallv().

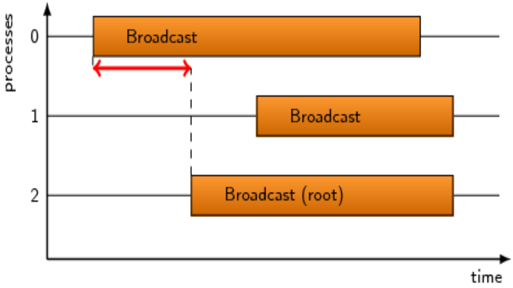


Note that the time reported by this pattern is not necessarily completely waiting time since some processes could -- at least theoretically -- already communicate with each other while others have not yet entered the operation.

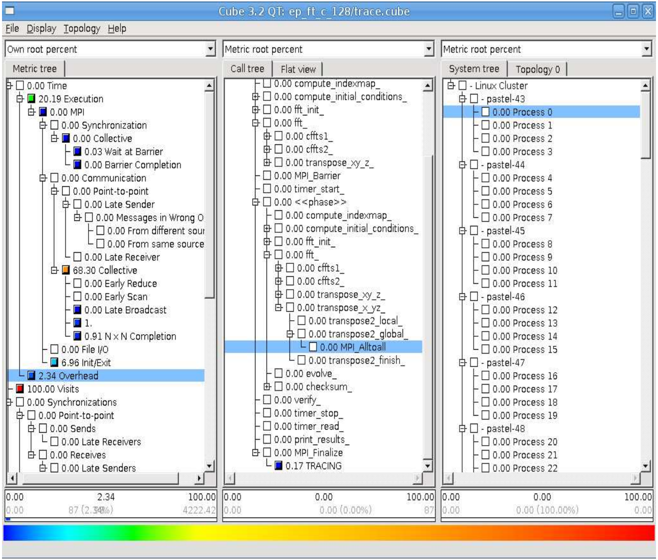
Demo - Late communication IV

Late Broadcast Time

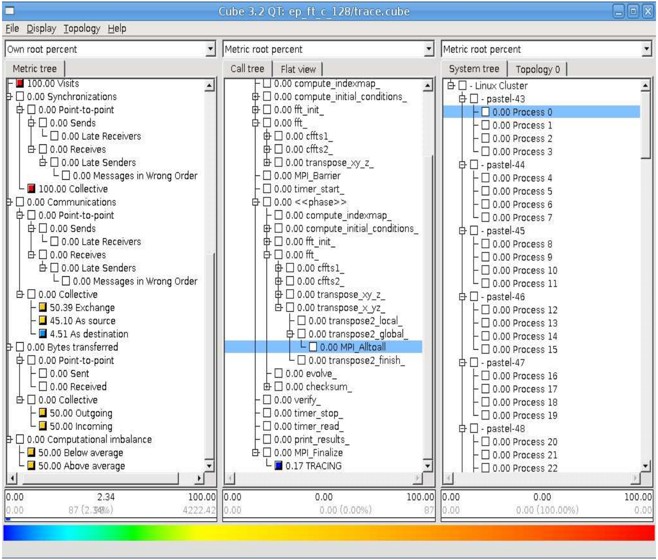
Description:
Collective communication operations that send data from one source process to all processes (i.e., 1-to-n) may suffer from waiting times if destination processes enter the operation earlier than the source process, that is, before any data could have been sent. The pattern refers to the time lost as a result of this situation. It applies to the MPI calls MPI_Bcast(), MPI_Scatter() and MPI_Scatterv().



Demo - Overhead



Demo - Events


Cube 3.2 QT: ep_ft_c_128/trace.cube
File Display Topology Help
Own root percent
Metric root percent
Metric root percent
Metric tree
Call tree
Flat view
System tree
Topology 0
100.00 Visits
0.00 Synchronizations
0.00 Point-to-point
0.00 Sends
0.00 Late Receivers
0.00 Receives
0.00 Late Senders
0.00 Messages in Wrong Order
100.00 Collective
0.00 Communications
0.00 Point-to-point
0.00 Sends
0.00 Late Receivers
0.00 Receives
0.00 Late Senders
0.00 Messages in Wrong Order
0.00 Collective
50.39 Exchange
45.10 As source
4.51 As destination
0.00 Bytes transferred
0.00 Point-to-point
0.00 Sent
0.00 Received
0.00 Collective
50.00 Outgoing
50.00 Incoming
0.00 Computational imbalance
50.00 Below average
50.00 Above average
0.00 compute_indexmap_
0.00 compute_initial_conditions_
0.00 fft_init_
0.00 fft_
0.00 cffts1_
0.00 cffts2_
0.00 transpose_xy_z_
0.00 MPI_Barrier
0.00 timer_start_
0.00 <<phase>>
0.00 compute_indexmap_
0.00 compute_initial_conditions_
0.00 fft_init_
0.00 fft_
0.00 cffts1_
0.00 cffts2_
0.00 transpose_xy_z_
0.00 transpose_x_yz_
0.00 transpose2_local_
0.00 transpose2_global_
0.00 MPI_Alltoall
0.00 transpose2_finish_
0.00 evolve_
0.00 checksum_
0.00 verify_
0.00 timer_stop_
0.00 timer_read_
0.00 print_results_
0.00 MPI_Finalize
0.17 TRACING
- Linux Cluster
- pastel-43
0.00 Process 0
0.00 Process 1
0.00 Process 2
0.00 Process 3
- pastel-44
0.00 Process 4
0.00 Process 5
0.00 Process 6
0.00 Process 7
- pastel-45
0.00 Process 8
0.00 Process 9
0.00 Process 10
0.00 Process 11
- pastel-46
0.00 Process 12
0.00 Process 13
0.00 Process 14
0.00 Process 15
- pastel-47
0.00 Process 16
0.00 Process 17
0.00 Process 18
0.00 Process 19
- pastel-48
0.00 Process 20
0.00 Process 21
0.00 Process 22
0.00 2.34 100.00
0.00 87 4222.42
0.00 0.00 100.00
0.00 0.00 (0.00%) 87
0.00 0.00 100.00
0.00 0.00 (100.00%) 0.00

EPIK measurement system

- Many configuration flags
- Declare buffer size for tracing files
- Configuration file epik.conf

EPIK user instrumentation API

- #include “epik_user.h”
- EPIK_USER_REG(phase,“ $<< phase >>$ “)
- EPIK_USER_START(phase)
- EPIK_USER_END(phase)

Cube3 utilities

- the command cube3_diff name1.cube name2.cube , creates a diff.cube file with the difference of both cube files.
- cube3_cut -r $' << phase >>'$ name.cube creates a cube with the phase sub-tree.

Pros and Cons

- Pros
  - Parallel analyzer SCOUT
  - Automatic recognition of bottlenecks
  - A new release will be available during SC '09 with many upgrades.
  - Very good structure of the output file format.
- Cons
  - Limited selective instrumentation
  - No Communication Matrix

Overhead comparison of Scalasca and TAU about profiling

	NAS Parallel Benchmarks	
	Overhead (%)	
Benchmark, Class, Cpus	TAU	Scalasca
FT, C, 32	5.02	2.17
FT, C, 64	5.5	2.37
FT, C, 128	9.45	8.08
FT, D, 128	1.03	0.79
EP, D, 32	0.29	0.2
EP, D, 64	1.79	1.2
EP, D, 128	3.7	2.82
EP, D, 192	5.44	3.8
DT, B, SH, 192	8.16	3.5

Overhead comparison of Scalasca and TAU about profiling II

	NAS Parallel Benchmarks	
	Overhead (%)	
Benchmark, Class, Cpus	TAU	Scalasca
DT, C, BH, 85	0.37	0.18
DT, C, WH, 85	0.97	0.21
LU, C, 32	3.46	1.64
LU, C, 64	4.51	4.39
LU, C, 128	7.02	3.05

References I

1. mpiP, TAU, Scalasca, on Blue Gene, Prashobh Balansundaram, IBM Dublin Software Labs, 2008

2. Technical Report, Profiling Tools for MPI Programming, J.V. Ashby, January 2008

3. PerfSuite: An Accessible, Open Source Performance Analysis Environment for Linux, Rick Kufrin, National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, 2005

4. A Scalable Cross-Platform Infrastructure for Application Performace Tunning Using Hardware Counters, S. Browne, J. Dongarra, N. Garner, K. London, P. Mucci

References II

5. MPI Performance Analysis Tools on Blue Gene/L, I-Hsin Chung, Robert E. Walkup, Hui-Fang Wen, Hao Yu.

6. Review of Performance Analysis Tools for MPI Parallel Programs, Shirley Moore, David Cronk, Kevin London, Jack Dongarra.

7. Workload Characterization Using the TAU Performance System.

8. The TAU Parallel Performance System, Sammer S. Shende, Allen D. Malony, University of Oregon, The International Journal of High Performance Computing Applications, Summer 2006, pp. 287-311

References III

9. Performance Instrumentation and Measurement for Terascale Systems, Jack Dongarra, Allen D. Malony, Shirley Moore, Philip Mucci, Sameer Shende.

10. The NAS Parallel Benchmarks, NAS Technical Report RNR-94-007, NASA Ames Research Center, (March 1994)

11. Review of Performance Analysis Tools for MPI Parallel Programs, Shirley Moore, David Cronk, Kevin London, Jack Dongarra

12. NAS Parallel Benchmarks version 2.4, Rob F. Van der Wijngaart, Computer Sciences Corporation, NASA Advanced Supercomputing (NAS) Division.

Thank you for your attention!

Questions?

Merci de votre attention!