MPI program tracing with Intel Trace Analyzer/Collector (ITAC)

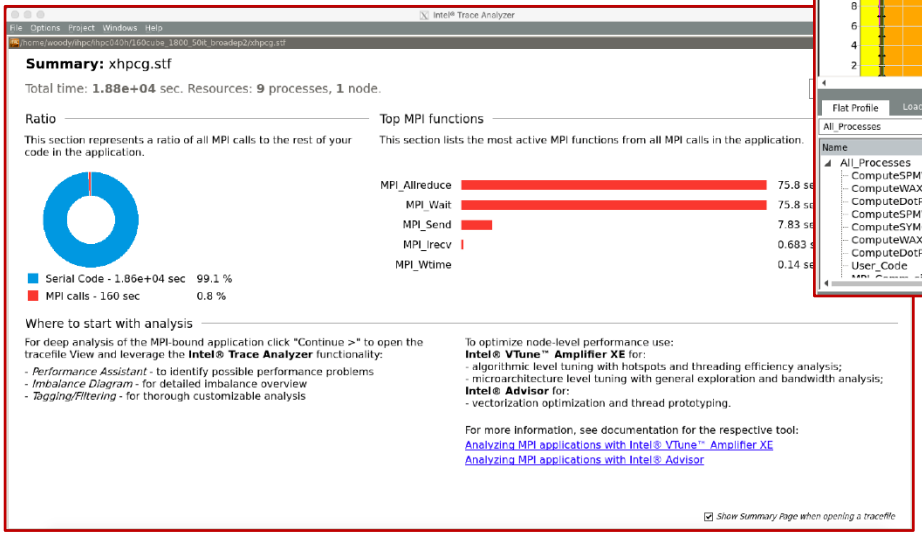
Georg Hager, Alireza Ghasemi, Ayesha Afzal

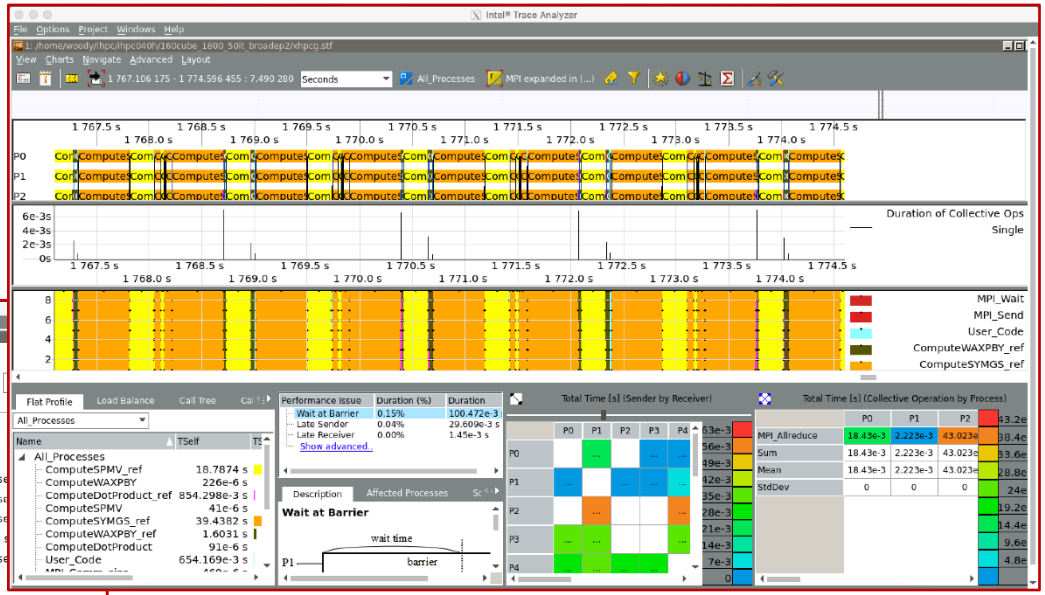
High Performance Computing

Intel Trace Anayzer and Collector

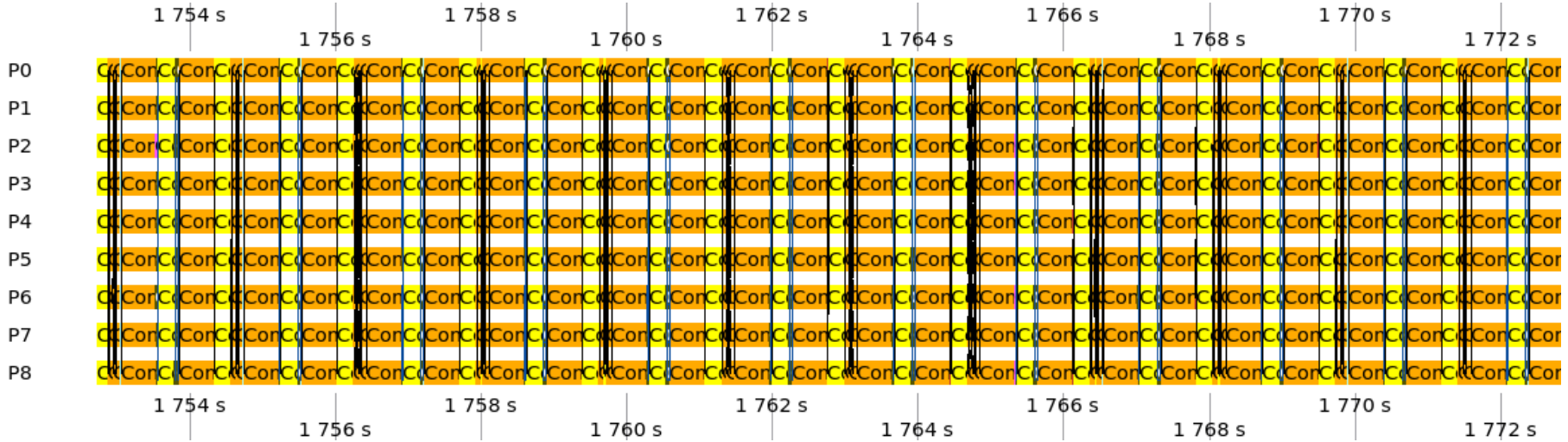
Event-based tool recording user function calls and MPI communication calls

GUI for advanced visualization



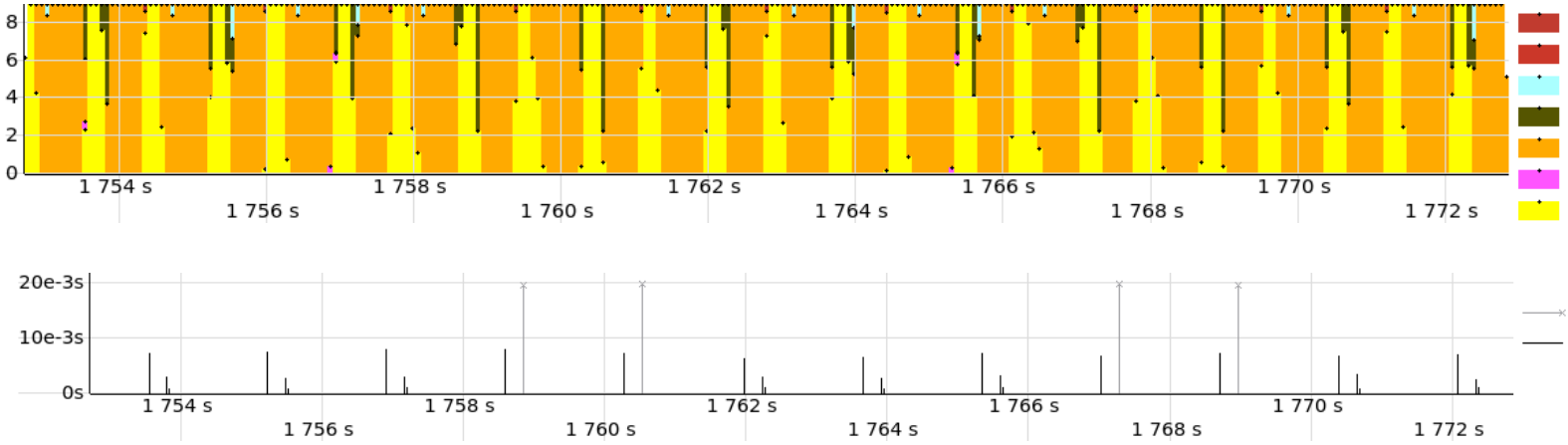


Event timeline view



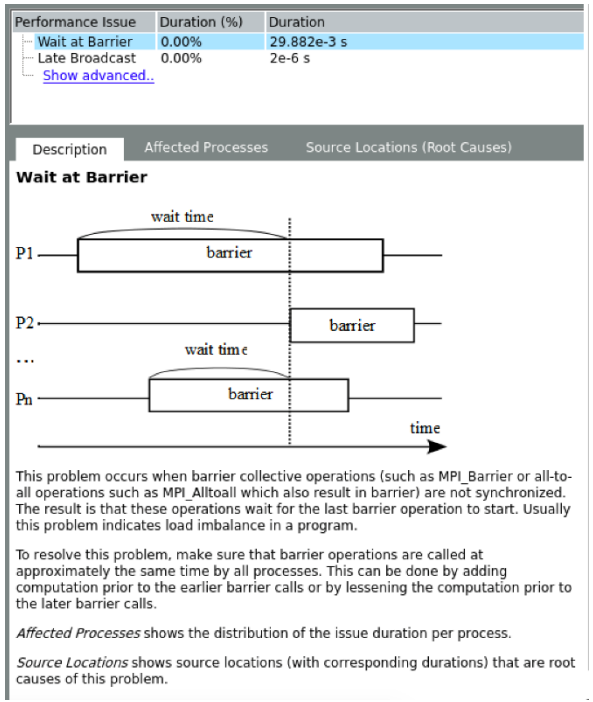
- Timeline of MPI and user function execution
- Message visualization
- Context menu provides details on functions/messages
- Zoom/pan

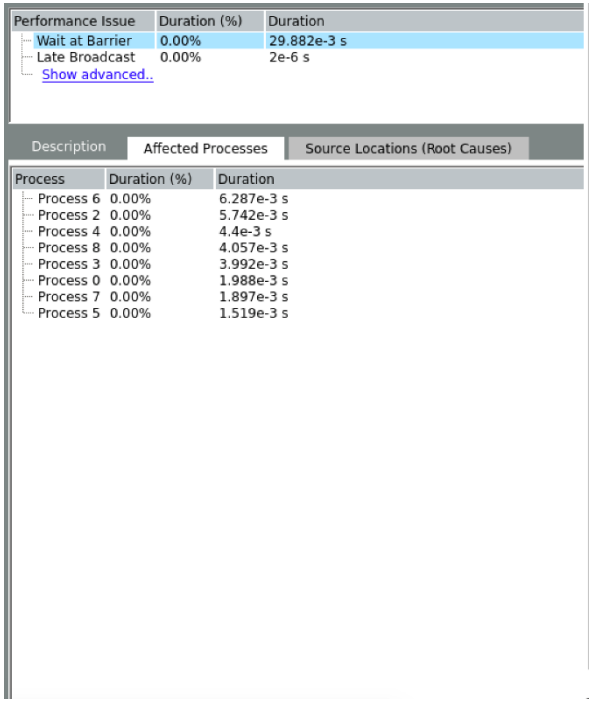
Quantitative and qualitative timelines



- Time spent in different MPI/user functions across processes
- Duration of certain things (collectives, PtP)

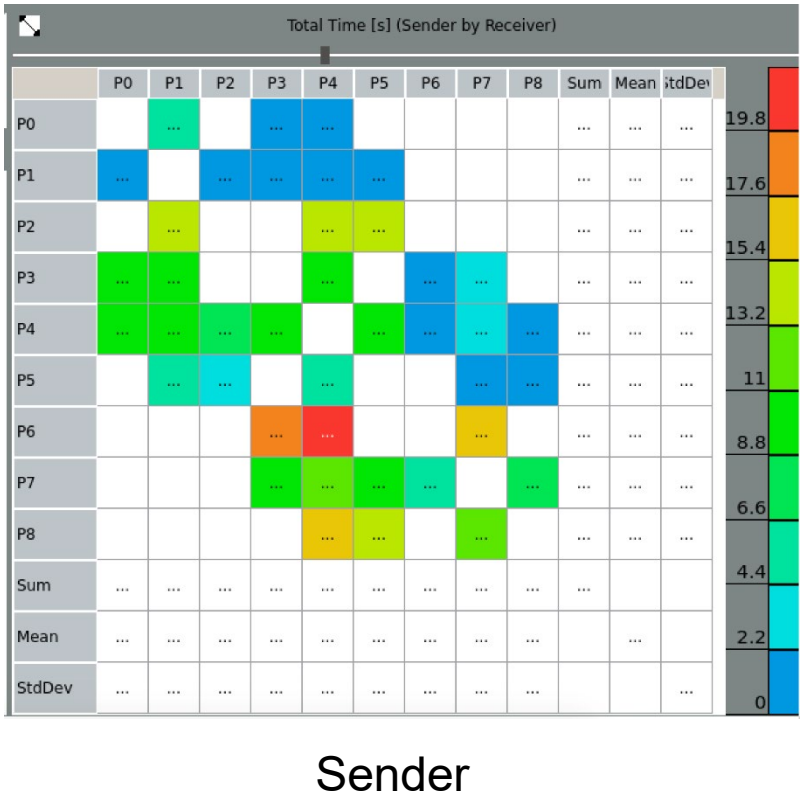
Performance advice





Context-sensitive advice on typical performance patterns

Message profile

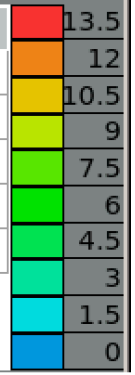


- Who sends how much to whom?
- How long does it take?
- Effective bandwidth?

Collective operations profile

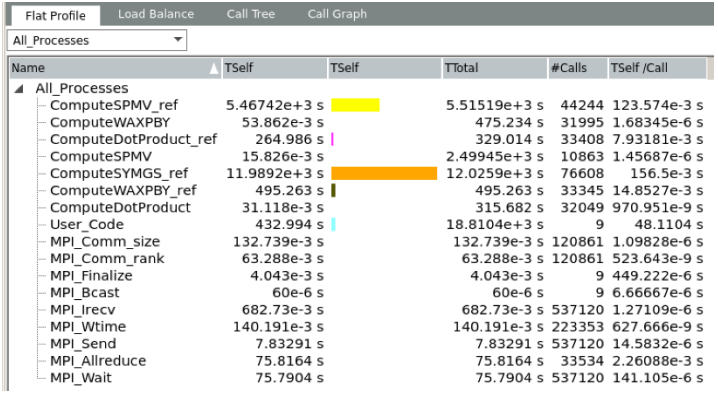
Total Time [s] (Collective Operation by Process)

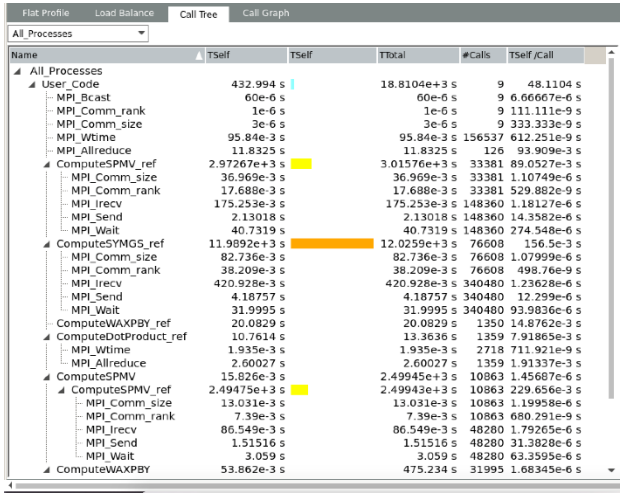
	P0	P1	P2	P3	P4	P5	P6	P7	P8	Sum	Mean	StdDev
MPI_Bcast	5e-6	7e-6	7e-6	7e-6	7e-6	7e-6	6e-6	7e-6	7e-6	60e-6	6.66667e-6	666.667e-9
MPI_Allreduce	6.98827	2.41008	14.1332	9.46671	9.80818	2.28141	12.1689	7.89127	10.6684	75.8164	8.42405	3.81376
Sum	6.98828	2.41009	14.1332	9.46671	9.80818	2.28142	12.1689	7.89127	10.6684	75.8165		
Mean	3.49414	1.20504	7.06659	4.73336	4.90409	1.14071	6.08444	3.94564	5.33422		4.21203	
StdDev	3.49413	1.20504	7.06658	4.73335	4.90409	1.1407	6.08444	3.94563	5.33422			5.00135

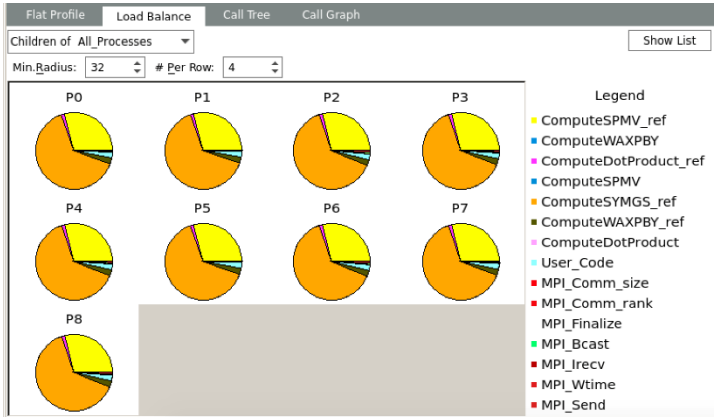


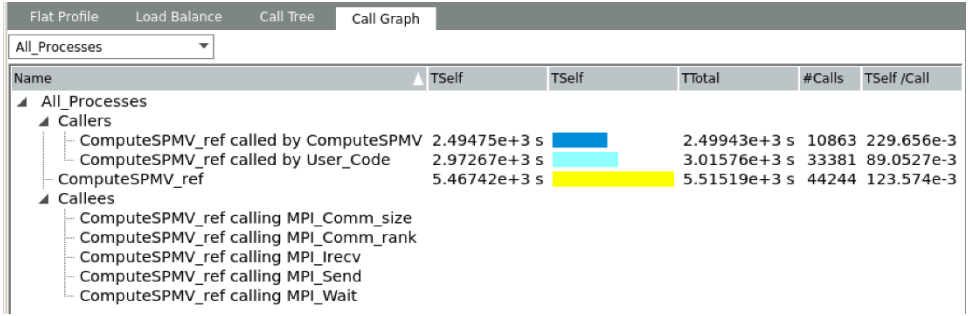
- Time spent in collective call
- Data volume sent/received

Functions profile, call tree/graph, load imbalance









Options for taking traces

- Caveat: Tracing can generate vast amounts of data!
- Compiler switches (only works with legacy Intel compiler and wrappers [mpiicc, mpiicpc, mpiifort])
  - `-trace` # record MPI calls (also possible with mpirun/mpiexec)
  - `-tcollect -trace` # record MPI and user code function calls
    # potential of large overhead and large trace size
  - `-tcollect-filter=func.txt -tcollect -trace` # filter file

func.txt example

```
'.*' OFF
'.*ComputeDotProduct.*' ON
'.*ComputeSYMGS.*' ON
'.*ComputeSPMV.*' ON
'.*ComputeWAXPBY.*' ON
```

More (important) configuration options

Environment variable	Default	Description
VT_FLUSH_PREFIX	... depends	control directly for temporary flush files
VT_LOGFILE_PREFIX	current working directory	control directly for physical trace information files
VT_LOGFILE_FORMAT	STF	`SINGLESTF`: rolls all trace files into one file (.single.stf)
VT_LOGFILE_NAME	${binary}.stf	control the name for the trace file
VT_MEM_BLOCKSIZE	64 KB	trace data in chunks of main memory
VT_MEM_FLUSHBLOCKS	1024	flushing is started when the number of blocks in memory exceeds this threshold
VT_MEM_MAXBLOCKS	1024	maximum number of blocks in main memory, if exceed the application is stopped until AUTOFLUSH/ MEM-OVERWRITE/ stop recording trace info
VT_CONFIG_RANK	0	control the process that reads and parses the configuration file

- Avoid rapid-fire dumping trace data into shared filesystems!
- Your fellow cluster users will hate you for it.

Alternatives

- ITAC is deprecated by Intel and will not be further developed (as of 2025)
  - Intel recommends VTune as a replacement, but this is not competitive

- Other tools with similar functionality
  - Vampir (commercial, scalable) https://vampir.eu/
  - Scalasca (for highly scalable programs, no trace view) https://www.scalasca.org/
  - Paraver https://tools.bsc.es/paraver

  - Jumpshot
    Don’t even bother.