Erlangen Regional Computing Center

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

ITAC: Intel Trace Collector and Analyzer for Parallel Computing

Ayesha Afzal[1], Georg Hager[1], Gerhard Wellein[1,2]

Erlangen, July 14, 2020

KONWIHR

HPC High Performance Computing

[1] Erlangen Regional Computing Center (RRZE)
[2] Department of Computer Science, University of Erlangen-Nürnberg

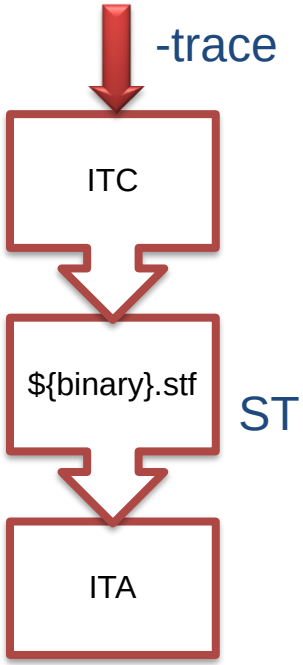
ITAC: profiling with Intel Trace Collector and Analyzer

Environment preparation

```
$ module load itac/2019up04 intel64/19.0up05
  (intelmpi/2019up05-intel)
```

Collector: compilation, linking and run

```
$ mpiicc -trace myProg.c -o ${binary}
$ mpirun -n 4 ./${binary}
```



format

STF: Structured Trace File

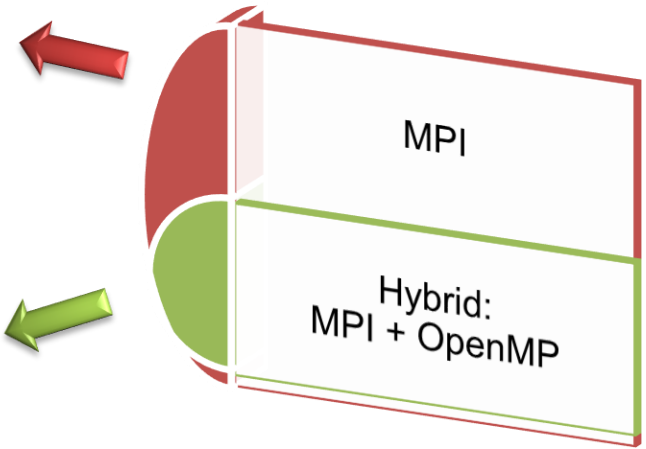
Analyzer

```
$ traceanalyzer ${binary}.stf &
```

Applications

Standard HPCG
https://www.hpcg-benchmark.org
C++ program
(Cascade lake SP/ Broadwell EP)

Ab initio molecular dynamics CPMD
https:// www.cpmd.org
Fortran program
(Broadwell EP and fat-tree
Omni-Path node interconnect)

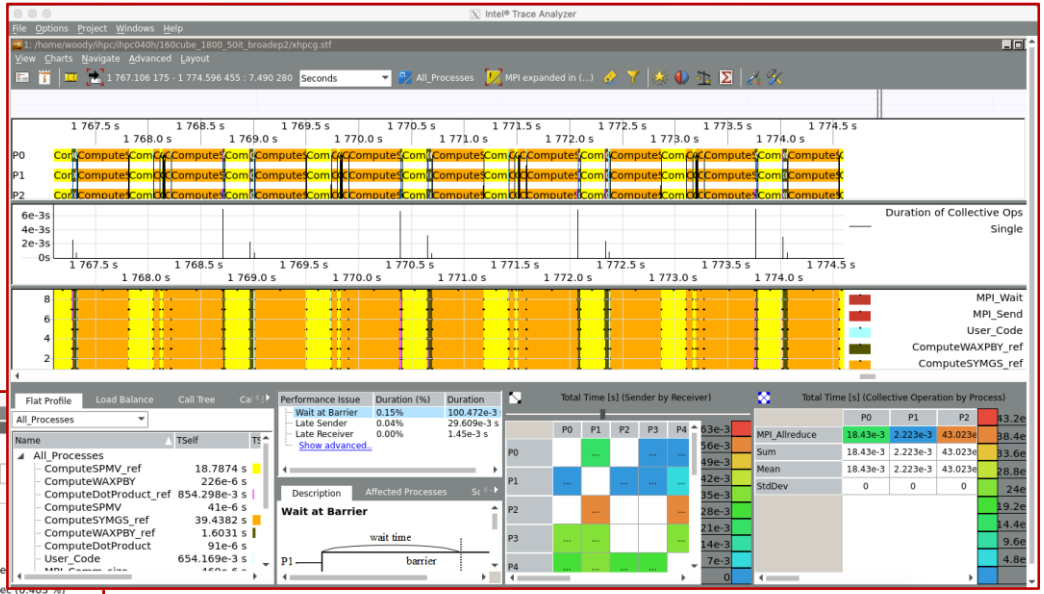


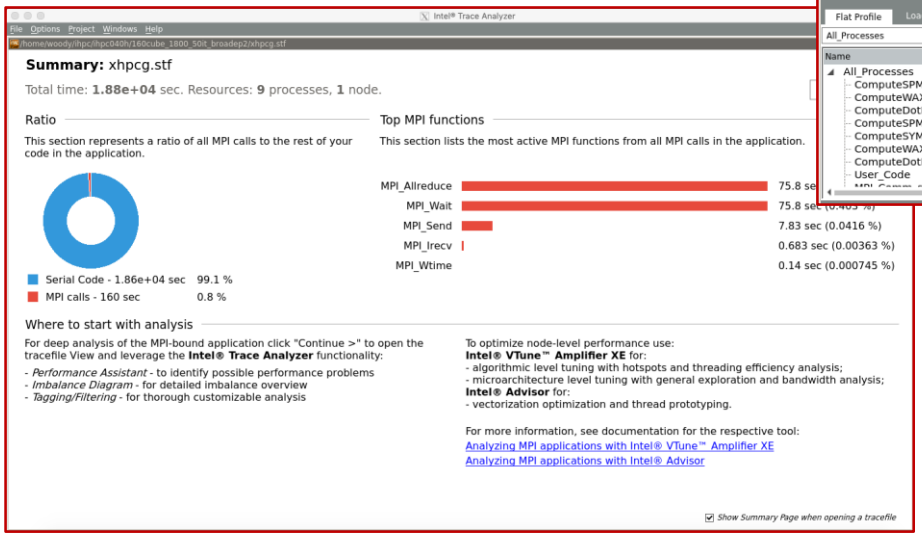
ITAC basic features

ITAC features: profiling with graphical visualization and statistical data

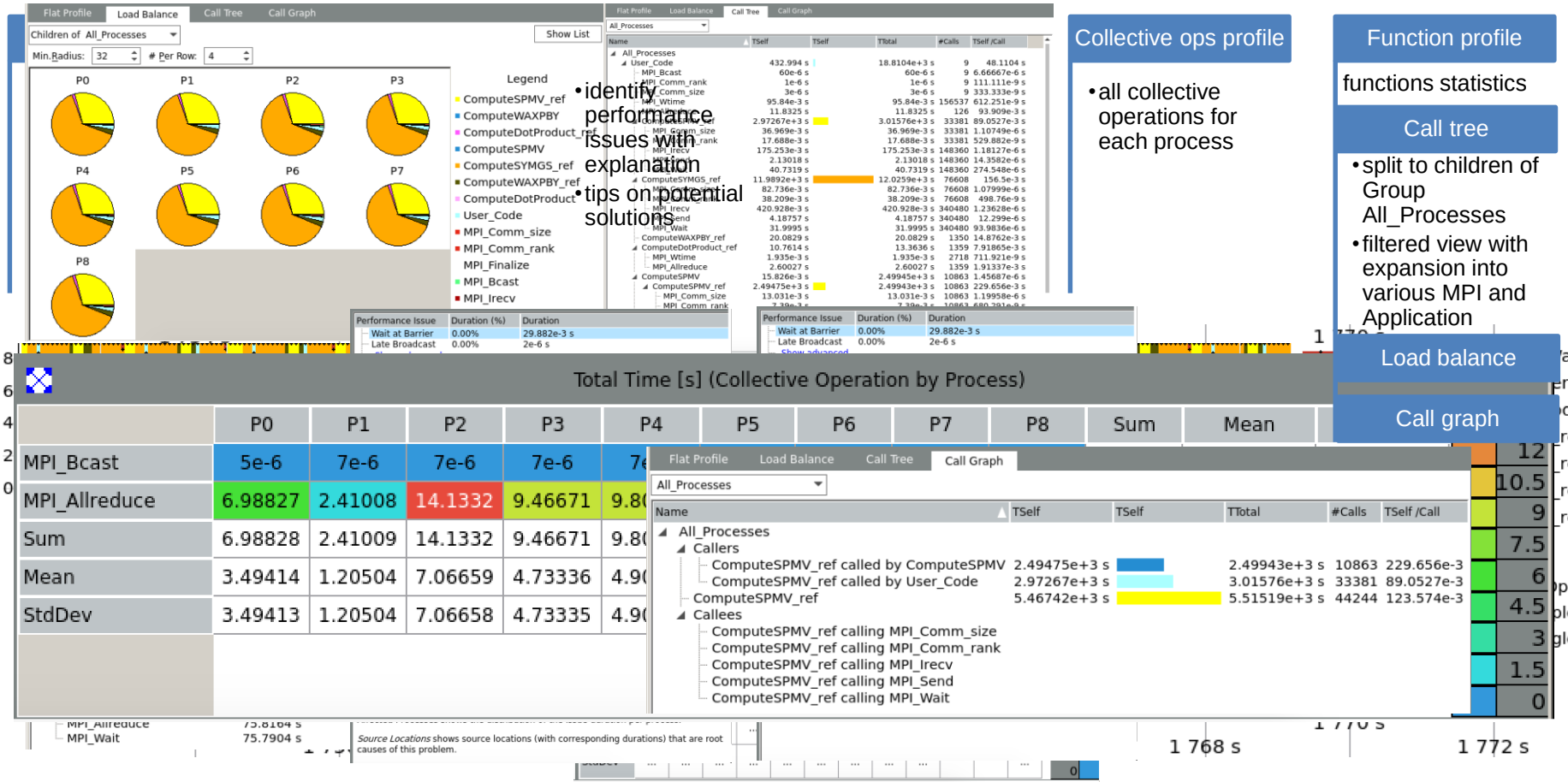
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Event-based approach that record
- user function calls
- MPI communication calls





ITAC features: profiling with graphical visualization and statistical data



- identify performance issues with explanation
- tips on potential solutions

Collective ops profile

- all collective operations for each process

Function profile

functions statistics

Call tree

- split to children of Group All_Processes
- filtered view with expansion into various MPI and Application

Load balance

Call graph

ITAC

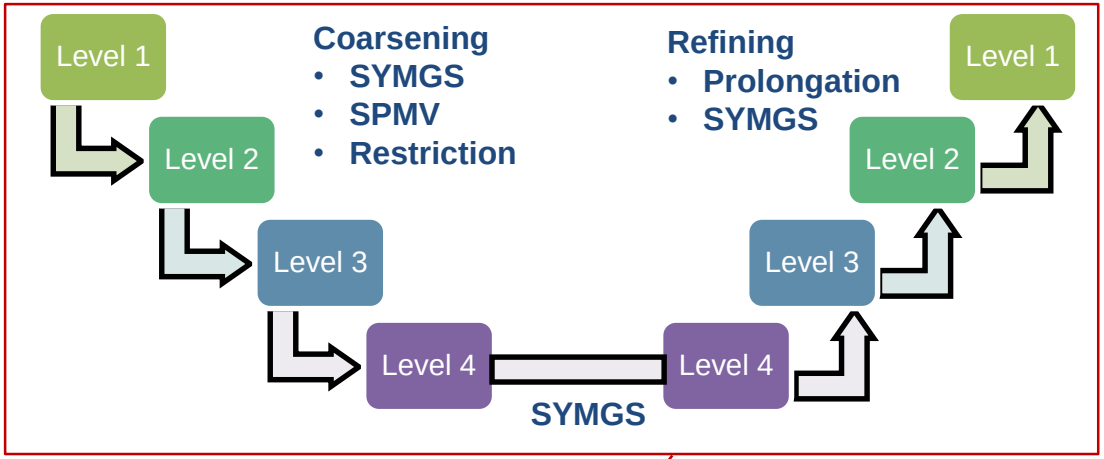
event timeline visualization

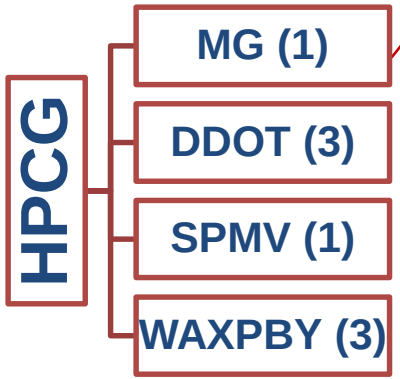
Algorithmic overview: preconditioned Conjugate Gradient

$\vec{p}_0 \leftarrow \vec{x}_0, \qquad \vec{r}_0 \leftarrow \vec{b} - A\vec{p}_0$

for $i = 1,2,$ *to* *max_iterations* **do**

- $\vec{z}_i \leftarrow M^{-1}\vec{r}_{i-1}$
- **if** $i = 1$ **then**
  - $\vec{p}_i \leftarrow \vec{z}_i$
  - $\alpha_i \leftarrow \text{dot\_prod}(\vec{r}_{i-1}, \vec{z}_i)$
- **else**
  - $\alpha_i \leftarrow \text{dot\_prod}(\vec{r}_{i-1}, \vec{z}_i)$
  - $\beta_i \leftarrow \alpha_i / \alpha_{i-1}$
  - $p_i \leftarrow \beta_i \vec{p}_{i-1} + \vec{z}_i$
- $\alpha_i \leftarrow \text{dot\_prod}(\vec{r}_{i-1}, \vec{z}_i) / \text{dot\_prod}(\vec{p}_i, A\vec{p}_i)$
- $\vec{x}_{i+1} \leftarrow \vec{x}_i + \alpha_i \vec{p}_i$
- $\vec{r}_i \leftarrow \vec{r}_{i-1} - \alpha_i A\vec{p}_i$
- **if** $\|\vec{r}_i\|_2 <$ *tolerance* **then**
  - STOP

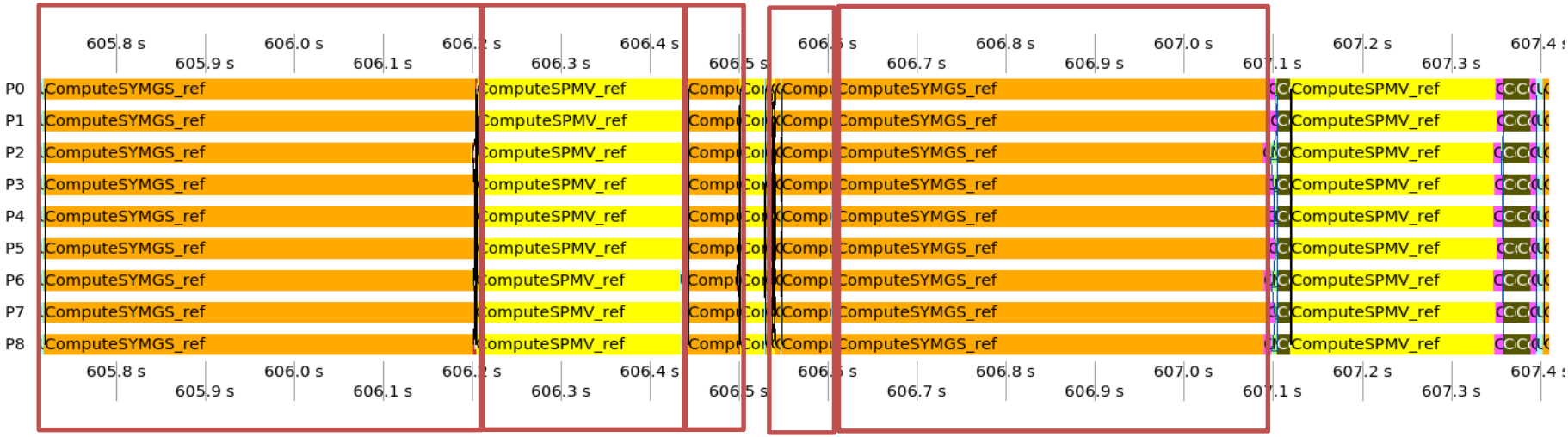




ITAC event timeline: MPI-parallelizable preconditioned CG structure

12 user-defined events + MPI communication

- SYMGS1+SPMV1+restriction+prolongation+SYMGS2

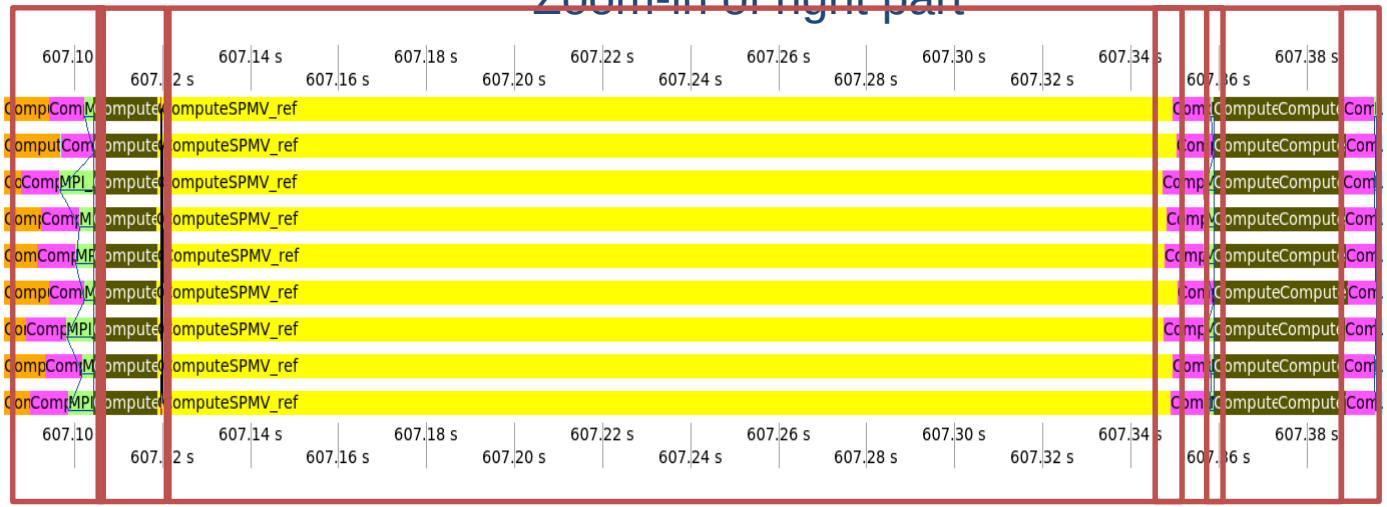


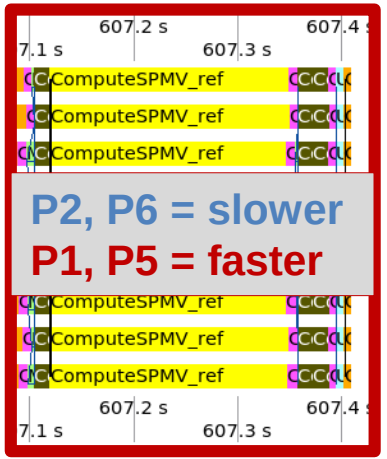
160^3 problem size , 1800 second run, Broadwell EP, one socket, 9 processes, 3 GB memory per process

ITAC event timeline: MPI-parallelizable preconditioned CG structure

Broadwell EP	Min (ms)	Max (ms)	Mean (ms)	Median (ms)	Variance (ms)	Skewness (ms)	Kurtosis (ms)	SD (ms)	IQR=Q3-Q1 (ms)
DDOT1	7.355	9.045	8.398	8.474	0.272	-0.913	0.697	0.521	0.663
DDOT2	8.300	9.270	8.793	8.713	0.109	-0.080	-0.880	0.330	0.305
DDOT3	6.543	6.890	6.725	6.688	0.013	0.129	-0.607	0.112	0.139

Zoom-in of right part

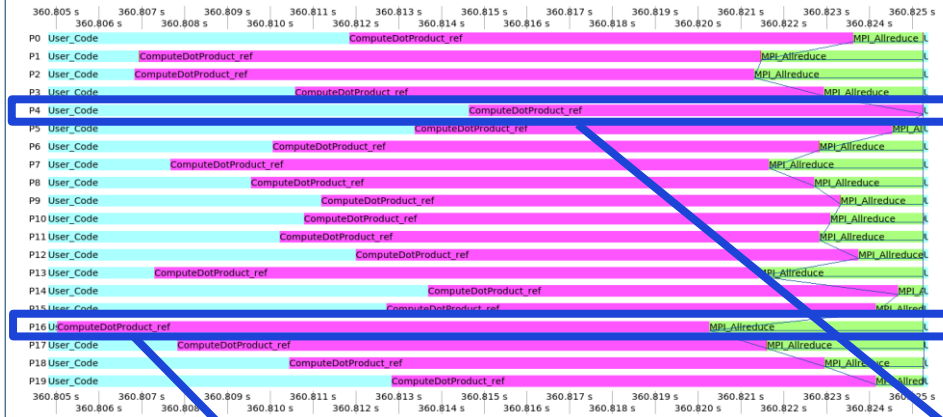


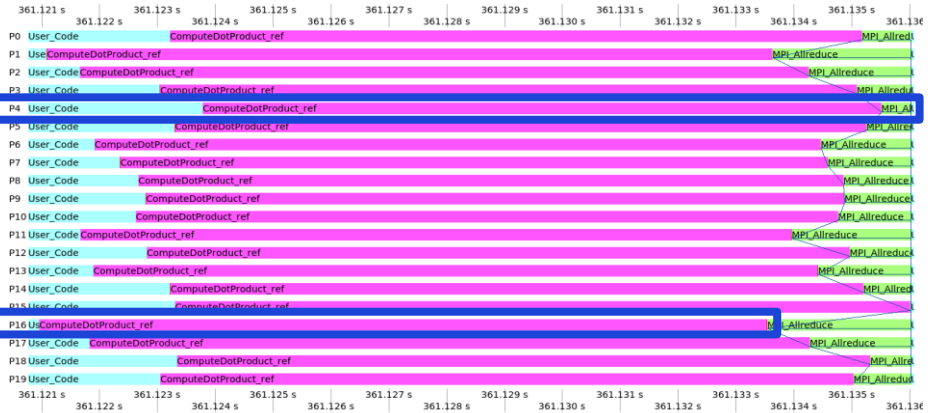


160^3 problem size , 1800 second run, Broadwell EP, one socket, 9 processes, 3 GB memory per process

ITAC event timeline: preconditioned CG on Cascade Lake SP

Cascade Lake EP	Min (ms)	Max (ms)	Mean (ms)	Median (ms)	Variance (ms)	Skewness (ms)	Kurtosis (ms)	SD (ms)	IQR=Q3-Q1 (ms)
DDOT1	10.592	15.225	12.635	12.426	1.761	0.372	-0.880	1.327	2.162
DDOT2	11.728	12.577	12.233	12.170	0.078	0.156	-1.142	0.279	0.536
DDOT3	7.603	8.008	7.723	7.686	0.014	1.364	0.909	0.116	0.111





DDOT1: Zoom-in

DDOT2: Zoom-in

P16 = slowest

P4 = fastest

ITAC

usage with cautions

ITAC usage with cautions: compiler switches, API

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

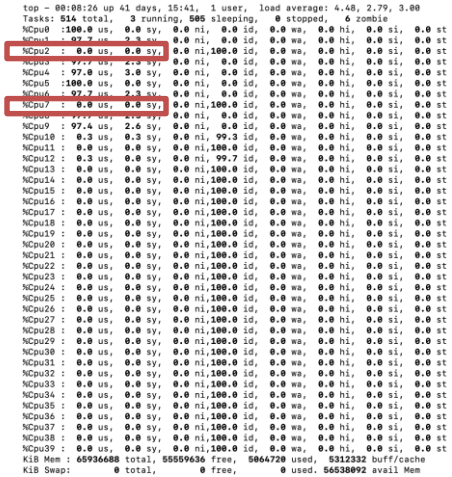
Be careful

check pinning correctness

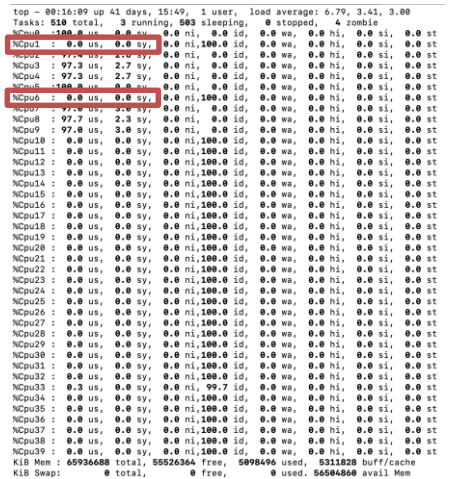
$ impi_info
$ export I_MPI_DEBUG=4

Example
$ export OMP_NUM_THREADS=5
$likwid-mpirun -np 2 -pin S0:0-4_S0:5-9 ./${binary}

"add -s 0x1"



17<intelmpi<19.0up02
problem with OMP T3



19.0up02 <intelmpi
problem with OMP T2

Metric	e0438:0:0	e0438:0:1	e0438:0:2	e0438:0:3	e0438:0:4	e0438:1:5	e0438:1:6	e0438:1:7	e0438:1:8	e0438:1:9
Runtime (RDTSC) [s]	76.4479	76.4479	76.4479	76.4479	76.4479	76.4635	76.4635	76.4635	76.4635	76.4635
Runtime unhalted [s]	75.1577	0.0065	71.0678	71.0700	71.0805	75.1883	0.0158	71.0499	70.9940	71.0678
Clock [MHz]	2200.0517	2200.0337	2200.0871	2200.0701	2200.0885	2200.0553	2200.0552	2200.0726	2200.0888	2200.0721
CPI	3.3481	0.7196	1.6558	1.6578	1.6571	3.3520	1.2420	1.6499	1.6614	1.6565
Memory read bandwidth [MBytes/s]	15219.1748	0	0	0	0	0	0	0	0	0
Memory read data volume [GBytes]	1163.4741	0	0	0	0	0	0	0	0	0
Memory write bandwidth [MBytes/s]	6125.8166	0	0	0	0	0	0	0	0	0
Memory write data volume [GBytes]	468.3059	0	0	0	0	0	0	0	0	0
Memory bandwidth [MBytes/s]	21344.9914	0	0	0	0	0	0	0	0	0
Memory data volume [GBytes]	1631.7800	0	0	0	0	0	0	0	0	0

ITAC usage with cautions: compiler switches, API

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Be careful

check pinning correctness ✓

$ impi_info

$ export I_MPI_DEBUG=4

use filtering options for large problems ✓

If not handled carefully,
generates loads of unrequired data

Reduce startup time

$ traceanalyzer --cli ${binary}.stf -c0 –w

VT API

Manually instrument the code to profile only interesting parts of an application or a subset of iterations

- run without ”-trace”
- #include <VT.h>
- -I${VT_ROOT}/include
- inserting calls to VT_traceoff() and VT_traceon()
- VT_begin(mark), VT_end(mark)

Compiler switches

1. `-trace`
2. `-tcollect -trace`
   function profile at a high price of maximum overhead
3. `-tcollect-filter=func.txt -tcollect -trace`

   `$cat func.txt:` funcX on|off
4. `-qopt-report`
   generate a full list of file and function names that the compiler recognizes from the compilation unit
5. `-tcollect -qopt-report -qopt-report-phase=tcollect`
   generate an optimization report with tcollect information to get a list of the file or routine strings that can be matched by the regular expression filters

Binary instrumentation at runtime

1. `-trace-imbalance`
   trace only the MPI functions that cause application load imbalance (idle at some point of the application run)
2. `-trace-collectives`
   trace only about collectives operations
3. `-trace-pt2pt`
   trace only about point-to-point operations

ITAC usage with cautions: environment variables

Environment variables	Default	Description
VT_FLUSH_PREFIX	/tmp	control directly for temporary flush files
VT_LOGFILE_PREFIX	current working directory	control directly for physical trace information files
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_LOGFILE_FORMAT	STF	`SINGLESTF`: rolls all trace files into one file (.single.stf)
VT_LOGFILE_NAME	${binary}	control the name for the tracefile
VT_CONFIG_RANK	0	control the process that reads and parses the configuration file

https://software.intel.com/content/www/us/en/develop/documentation/itc-user-and-reference-guide/top/intel-trace-collector-reference/configuration-reference/configuration-options.html

Environment variables: set up in the
1. corresponding environment variables
2. command line when running your application
3. configuration file

$ export VT_CONFIG=<config_file>

```
# enable all Application activities
ACTIVITY Application ON

# disable all MPI activity
ACTIVITY MPI OFF

# enable all bcasts, barrier, allreduce, recvs and sends
SYMBOL MPI_WAITALL ON
SYMBOL MPI_IRECV ON
SYMBOL MPI_ISEND ON
SYMBOL MPI_BARRIER ON
SYMBOL MPI_ALLREDUCE ON
```

Use ITAC wisely with filtering options of your own preferences

ITAC demo

Title	ITAC: Intel Trace Collector and Analyzer for Parallel Computing
Contact	Ayesha Afzal ayesha.afzal@fau.de
Acknowledgement	Georg Hager, Gerhard Wellein

Thanks for Listening

Questions?

KONWIHR OMI4papps

Optimization, Modeling and Implementation for highly parallel applications

<http://www.konwihr.uni-erlangen.de/projekte/laufende-projekte/omi4papps.shtml>

This work was supported by KONWIHR, the Bavarian Competence Network for Scientific High Performance Computing, under project OMI4papps