Lecture - MPI Profiling and Tracing

HPC

Assoc.Prof. Dr. Sascha Hunold

TU Wien

2024-01-09

Overview

- performance analysis of parallel applications
  - profiling tools
  - tracing tools
- hardware performance counters
  - libraries

Recall Matrix Matrix Multiplication

Alg 0, version ijk

```
void mulijk(double** a, double** b, double** c, int m, int n, int p) {
  int i, j, k;
  for (i = 0; i < m; ++i)
    for (j = 0; j < p; ++j)
      for (k = 0; k < n; ++k)
        c[i][j] += a[i][k] * b[k][j];
}
```

Alg 1, version ikj

```
void mulikj(double** a, double** b, double** c, int m, int n, int p) {
  int i, j, k;
  for (i = 0; i < m; ++i)
    for (k = 0; k < n; ++k)
      for (j = 0; j < p; ++j)
        c[i][j] += a[i][k] * b[k][j];
}
```

Gprof

Gprof

- utility for timing subroutines, library calls, intrinsic functions
- code must be instrumented (-pg, symbol table)

```
hunold@hydra-head:~/tmp/mmm_sahu$ make mmm_sahu_gprof
gcc -Wall -O3 --std=c99 -L/opt/local/lib -pg -o mmm_sahu_gprof mmm_sahu.c
```

- now, we can obtain the profile and later inspect it using both the binary and the profile

```
hunold@hydra-head:~/tmp/mmm_sahu$ ./mmm_sahu_gprof -n 1000 -a 1
M=1000 N=1000 P=1000
ikj:  0.35 seconds
GFlops      : 5.682
hunold@hydra-head:~/tmp/mmm_sahu$ gprof mmm_sahu_gprof gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
100.27      0.35     0.35                             mulikj

 %         the percentage of the total running time of the
time       program used by this function.
```

Gprof - How does it work? I

- sample-based profiling
- sampling rate: 100 Hz sampling rate
- statistical sampling of program counter
- insert special monitor code around routines
  - execution counts
- infer execution time of routines by sampling PC (for time-sharing systems)
- two outputs possible
  1. call-graph
     - what functions were called by other functions
  2. flat profile
- for more information see: S. L. Graham, P. B. Kessler, and M. K. Mckusick. "Gprof: A Call Graph Execution Profiler". In: *Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction*. SIGPLAN '82. New York, NY, USA: ACM, 1982, pp. 120–126. ISBN: 0-89791-074-5. DOI: 10.1145/800230.806987

valgrind / (K)Cachegrind

valgrind

- `http://valgrind.org/`
- instrumentation framework for building dynamic analysis tools (GPLv2)
- no recompilation required
- dynamic binary translation of user-level processes
- tools
  - memcheck
    - mem leaks, faulty mem accesses
  - cachegrind
    - cache profiler
  - callgrind
    - extension to cachegrind + information about callgraphs
  - massif
    - heap profiler, regular snapshots of heap, usage over time
  - helgrind
    - thread debugger, finds races in multithreaded programs
  - drd
    - detecting errors in multithreaded C and C++ programs

Cachegrind

- simulates
  - program with cache
  - independent L1 instruction and data caches
  - unified L2 cache
- if more than 2 cache levels, simulate 1st and last level cache (as last level has biggest influence on runtime)
- refers to I1, D1, LL caches
- Cachegrind tries to auto-detect the cache configuration

Cachegrind - Matrix Multiplication Example I

- 2 versions
  - 0: `ijk`
  - 1: `ijk`

```
hunold@hydra-head:~/tmp/mmm_sahu$ ./mmm_sahu -n 1000 -a 0
M=1000 N=1000 P=1000
ijk:  1.28 seconds
GFlops      : 1.560
```

```
hunold@hydra-head:~/tmp/mmm_sahu$ ./mmm_sahu -n 1000 -a 1
M=1000 N=1000 P=1000
ikj:  0.37 seconds
GFlops      : 5.359
```

Cachegrind - Matrix Multiplication Example II

```
hunold@hydra-head:~/tmp/mmm_sahu$ valgrind --tool=cachegrind ./mmm_sahu -n 1000 -a 0
==3650832== Cachegrind, a cache and branch-prediction profiler
==3650832== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==3650832== Using Valgrind-3.16.1 and LibVEX; rerun with -h for copyright info
==3650832== Command: ./mmm_sahu -n 1000 -a 0
==3650832==
M=1000 N=1000 P=1000
ijk: 43.79 seconds
GFlops      : 0.046
==3650832==
==3650832== I   refs:      8,133,122,999
==3650832== I1  misses:            1,370
==3650832== LLi misses:            1,365
==3650832== I1  miss rate:          0.00%
==3650832== LLi miss rate:          0.00%
==3650832==
==3650832== D   refs:      4,061,071,374  (3,043,053,978 rd   + 1,018,017,396 wr)
==3650832== D1  misses:    1,250,381,227  (1,250,130,090 rd   + 251,137 wr)
==3650832== LLd misses:          378,762  (      127,746 rd   + 251,016 wr)
==3650832== D1  miss rate:          30.8% (         41.1%     + 0.0%  )
==3650832== LLd miss rate:           0.0% (          0.0%     + 0.0%  )
==3650832==
==3650832== LL refs:       1,250,382,597  (1,250,131,460 rd   + 251,137 wr)
==3650832== LL misses:           380,127  (      129,111 rd   + 251,016 wr)
==3650832== LL miss rate:            0.0% (          0.0%     + 0.0%  )
```

```
hunold@hydra-head:~/tmp/mmm_sahu$ valgrind --tool=cachegrind ./mmm_sahu -n 1000 -a 1
==3650778== Cachegrind, a cache and branch-prediction profiler
==3650778== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==3650778== Using Valgrind-3.16.1 and LibVEX; rerun with -h for copyright info
==3650778== Command: ./mmm_sahu -n 1000 -a 1
==3650778==
M=1000 N=1000 P=1000
ikj: 12.32 seconds
GFlops      : 0.162
==3650778==
==3650778== I   refs:      4,152,129,943
==3650778== I1  misses:            1,364
==3650778== LLi misses:            1,359
==3650778== I1  miss rate:          0.00%
==3650778== LLi miss rate:          0.00%
==3650778==
==3650778== D   refs:      1,564,078,358  (1,046,060,963 rd   + 518,017,395 wr)
==3650778== D1  misses:      125,631,226  (  125,380,090 rd   + 251,136 wr)
==3650778== LLd misses:          378,763  (      127,748 rd   + 251,015 wr)
==3650778== D1  miss rate:           8.0% (          12.0%     + 0.0%  )
==3650778== LLd miss rate:           0.0% (          0.0%     + 0.0%  )
==3650778==
==3650778== LL refs:         125,632,590  (  125,381,454 rd   + 251,136 wr)
==3650778== LL misses:           380,122  (      129,107 rd   + 251,015 wr)
==3650778== LL miss rate:            0.0% (          0.0%     + 0.0%  )
```

PAPI

PAPI - About

- `http://icl.cs.utk.edu/papi/`
- Performance Application Programming Interface (PAPI)
- PAPI project originated at the University of Tennessee's Innovative Computing Laboratory
- "design, standardize, and implement a portable and efficient API to access the hardware performance counters found on modern microprocessors"

PAPI - papi_avail

```
hunold@hydra-head:~/tmp/papi/papi_mm$ papi_avail
Available PAPI preset and user defined events plus hardware information.
--------------------------------------------------------------------------------
PAPI version             : 6.0.0.1
Operating system         : Linux 5.10.0-18-amd64
Vendor string and code   : GenuineIntel (1, 0x1)
Model string and code    : Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz (85, 0x55)
CPU revision             : 4.000000
CPUID                    : Family/Model/Stepping 6/85/4, 0x06/0x55/0x04
CPU Max MHz              : 3700
CPU Min MHz              : 1000
...
--------------------------------------------------------------------------------

================================================================================
  PAPI Preset Events
================================================================================
    Name        Code     Avail Deriv Description (Note)
PAPI_L1_DCM  0x80000000  Yes   No   Level 1 data cache misses
PAPI_L1_ICM  0x80000001  Yes   No   Level 1 instruction cache misses
PAPI_L2_DCM  0x80000002  Yes   Yes  Level 2 data cache misses
PAPI_L2_ICM  0x80000003  Yes   No   Level 2 instruction cache misses
PAPI_L3_DCM  0x80000004  No    No   Level 3 data cache misses
PAPI_L3_ICM  0x80000005  No    No   Level 3 instruction cache misses
PAPI_L1_TCM  0x80000006  Yes   Yes  Level 1 cache misses
PAPI_L2_TCM  0x80000007  Yes   No   Level 2 cache misses
PAPI_L3_TCM  0x80000008  Yes   No   Level 3 cache misses
```

PAPI on hydra

- adapted (new PAPI API) matrix multiplication example from `http://docs.rcc.uchicago.edu/_downloads/matrixmult_papi.c`

```c
int main( int argc, char *argv[] ) {
  int i, j, k;
  double *A, *B, *C;
  int retval;

  A = (double *)malloc(N*M*sizeof(double));
  B = (double *)malloc(N*M*sizeof(double));
  C = (double *)malloc(N*N*sizeof(double));

  if ( A == NULL || B == NULL || C == NULL ) {
    fprintf(stderr,"Error allocating memory!\n");
    exit(1);
  }

  /* initialize A & B */
  for ( i = 0; i < N; i++ ) {
    for ( j = 0; j < M; j++ ) {
      A[M*i+j] = 3.0;
      B[N*j+i] = 2.0;
    }
  }
```

```c
  for ( i = 0; i < N*N; i++ ) {
    C[i] = 0.0;
  }

  retval = PAPI_hl_region_begin("computation_mul");
  if ( retval != PAPI_OK )
    ERROR_RETURN(retval);

  for ( i = 0; i < N; i++ )
    for ( j = 0; j < N; j++ )
      for ( k = 0; k < M; k++ )
        C[N*i+j] += A[M*i+k]*B[N*k+j];

  retval = PAPI_hl_read("computation_mul");
  if ( retval != PAPI_OK )
    ERROR_RETURN(retval);

  retval = PAPI_hl_region_end("computation_mul");
  if ( retval != PAPI_OK )
    ERROR_RETURN(retval);

  free(A);
  free(B);
```

PAPI on hydra I

- let's load PAPI from spack

```
module load papi-6.0.0.1-gcc-12.1.0-dhfk33b
```

- compile example using PAPI, note: set the `rpath` correctly

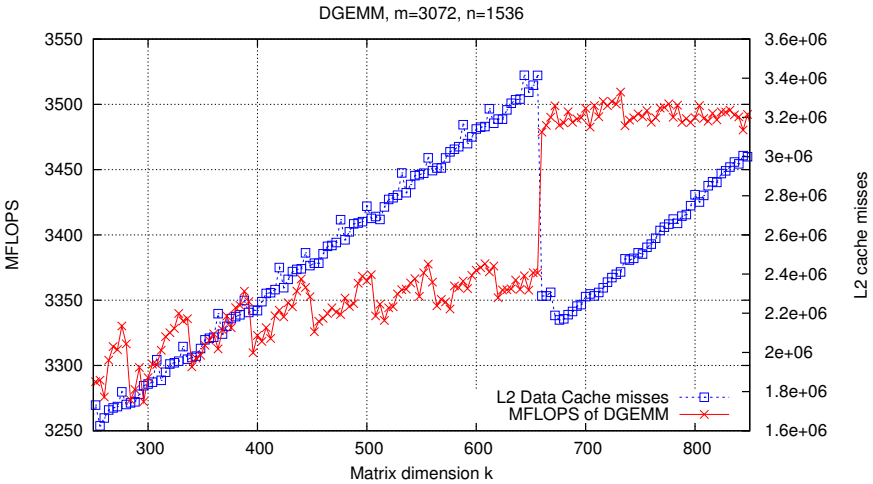
```
hunold@hydra-head:~/tmp/papi/papi_mm$ gcc
↪  -I/opt/spack/spack_git/opt/spack/linux-debian11-skylake_avx512/gcc-12.1.0/papi-6.0.0.1-dhfk33bbfxro4scbsavpxtl2cwzswlqf/include -o matrixmult_papi
↪  matrixmult_papi.c
↪  /opt/spack/spack_git/opt/spack/linux-debian11-skylake_avx512/gcc-12.1.0/papi-6.0.0.1-dhfk33bbfxro4scbsavpxtl2cwzswlqf/lib/libpapi.so
↪  -Wl,-rpath,/opt/spack/spack_git/opt/spack/linux-debian11-skylake_avx512/gcc-12.1.0/papi-6.0.0.1-dhfk33bbfxro4scbsavpxtl2cwzswlqf/lib/
```

PAPI on hydra II

- we are interested in the L2 and L3 cache misses

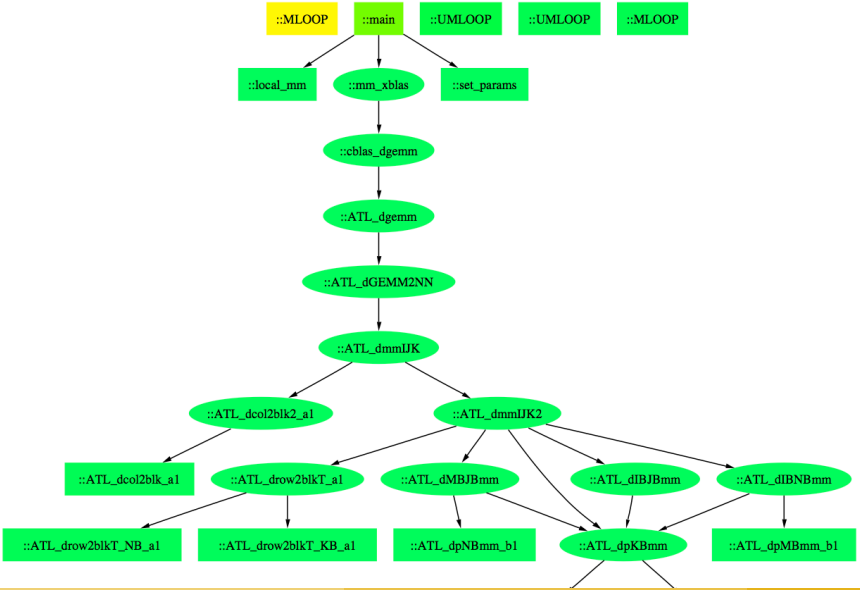
```
hunold@hydra-head:~/tmp/papi/papi_mm$ PAPI_EVENTS="PAPI_TOT_INS,PAPI_L2_TCM,PAPI_L3_TCM" PAPI_REPORT=1 ./matrixmult_papi
PAPI-HL Output:
{
  "cpu in mhz":"3700",
  "threads":[
    {
      "id":"3239574",
      "regions":[
        {
          "computation_mul":{
            "region_count":"1",
            "cycles":{
              "total":"3132803802",
              "read_1":"3132797108"
            },
            "PAPI_TOT_INS":{
              "total":"14760639347",
              "read_1":"14760636746"
            },
            "PAPI_L2_TCM":{
              "total":"60089333",
              "read_1":"60089139"
            },
            "PAPI_L3_TCM":{
              "total":"107588",
              "read_1":"107509"
            }
          }}}}}}
```

PAPI in Research

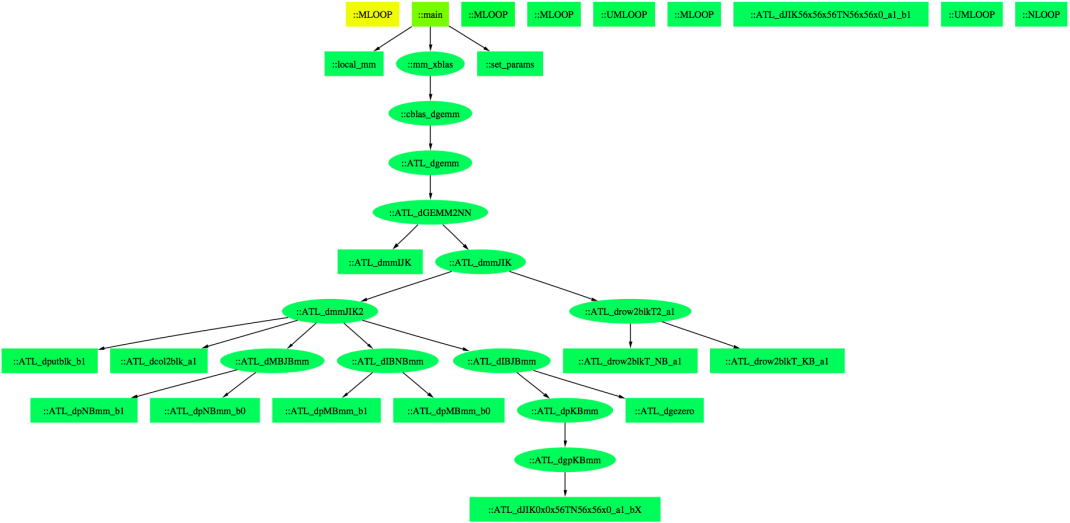


source: S. Hunold and T. Rauber. "Automatic Tuning of PDGEMM Towards Optimal Performance". In: *Euro-Par*. Ed. by J. C. Cunha and P. D. Medeiros. Vol. 3648. Lecture Notes in Computer Science. Springer, 2005, pp. 837–846. ISBN: 3-540-28700-0

ATLAS in detail, dimension k = 656, `gprof` trace



ATLAS in detail, dimension k = 672, `gprof` trace



perf

perf I

- available on virtually all Linux systems
- easy to inspect performance counters
- note: also needs correct permissions (often disabled by default)

```
hunold@hydra-head:~$ perf stat hostname
hydra-head

 Performance counter stats for 'hostname':

              1.09 msec task-clock                #    0.422 CPUs utilized
                 0      context-switches          #    0.000 K/sec
                 0      cpu-migrations            #    0.000 K/sec
                52      page-faults               #    0.048 M/sec
         1,084,187      cycles                    #    0.993 GHz
           915,036      instructions              #    0.84  insn per cycle
           184,107      branches                  #  168.554 M/sec
             7,553      branch-misses             #    4.10% of all branches

       0.002586291 seconds time elapsed

       0.002864000 seconds user
       0.000000000 seconds sys
```

perf II

- let us test it to the extreme
- `stream` is a DRAM benchmark, thus, we expect many cache misses

```
hunold@hydra-head:~/tmp/stream$ perf stat -e cache-references,cache-misses ./stream_omp
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 1000000000 (elements), Offset = 0 (elements)
Memory per array = 7629.4 MiB (= 7.5 GiB).
Total memory required = 22888.2 MiB (= 22.4 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 16
Number of Threads counted = 16
...
```

```
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           65889.2     0.243103     0.242832     0.243615
Scale:          65815.8     0.243272     0.243103     0.243532
Add:            74363.7     0.323089     0.322738     0.323343
Triad:          74316.7     0.323208     0.322942     0.323560
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------

 Performance counter stats for './stream_omp':

    13,444,515,011      cache-references
    13,336,303,146      cache-misses              #   99.195 % of all cache refs

      14.527950413 seconds time elapsed

     190.677097000 seconds user
      10.725886000 seconds sys
```

likwid

likwid

- likwid = *Like I knew what I am doing* (GPLv3)
- `https://github.com/RRZE-HPC/likwid`
- "It contains the following tools:
  - likwid-topology: Show the thread and cache topology
  - likwid-perfctr: Measure hardware performance counters on Intel and AMD processors
  - likwid-features: Show and Toggle hardware prefetch control bits on Intel Core 2 processors
  - likwid-pin: Pin your threaded application without touching your code (supports pthreads, Intel OpenMP and gcc OpenMP)
  - likwid-mpirun: Script enabling simple and flexible pinning of MPI and MPI/threaded hybrid applications
  - .. "

likwid - `likwid-perfctr`

- "`likwid-perfctr` supports the following modes:
  - wrapper mode: Use likwid-perfctr as a wrapper to your application. You can measure without altering your code.
  - stethoscope mode: Measure performance counters for a variable time duration independent of any code running.
  - timeline mode: Output performance metric in specified frequency (can be ms or s)
  - marker API: Only measure regions in your code, still `likwid-perfctr` controls what to measure."

source: `https://code.google.com/p/likwid/wiki/LikwidPerfCtr`

likwid on hydra / likwid-topology

```
hunold@hydra-head:~/tmp/stream$ likwid-topology
--------------------------------------------------------------------------------
CPU name:        Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz
CPU type:        Intel Skylake SP processor
CPU stepping:        4
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:                 1
Cores per socket:        16
Threads per core:        1
--------------------------------------------------------------------------------
HWThread        Thread        Core        Die        Socket        Available
0               0             0           0          0             *
1               0             1           0          0             *
..
--------------------------------------------------------------------------------
Socket 0:                ( 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 )
--------------------------------------------------------------------------------
********************************************************************************
Cache Topology
********************************************************************************
Level:                        1
Size:                         32 kB
Cache groups:                 ( 0 ) ( 1 ) ( 2 ) ( 3 ) ( 4 ) ( 5 ) ( 6 ) ( 7 ) ( 8 ) ( 9 ) ( 10 ) ( 11 ) ( 12 ) ( 13 ) ( 14 ) ( 15 )
--------------------------------------------------------------------------------
Level:                        2
Size:                         1 MB
Cache groups:                 ( 0 ) ( 1 ) ( 2 ) ( 3 ) ( 4 ) ( 5 ) ( 6 ) ( 7 ) ( 8 ) ( 9 ) ( 10 ) ( 11 ) ( 12 ) ( 13 ) ( 14 ) ( 15 )
--------------------------------------------------------------------------------
Level:                        3
Size:                         22 MB
Cache groups:                 ( 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 )
--------------------------------------------------------------------------------
********************************************************************************
NUMA Topology
```

likwid on hydra I

- use our example from PAPI, but remove the PAPI calls (important!)

```
hunold@hydra-head:~/tmp/papi/papi_mm$ gcc -o matrixmult_likwid matrixmult_likwid.c
```

- we now analyze the double precision floating point counts using `likwid`

```
hunold@hydra-head:~/tmp/papi/papi_mm$ likwid-perfctr -a
    Group name         Description
--------------------------------------------------------------------------------
        ENERGY         Power and Energy consumption
        CACHES         Cache bandwidth in MBytes/s
     FLOPS_AVX         Packed AVX MFLOP/s
CYCLE_ACTIVITY         Cycle Activities
        MEM_SP         Overview of arithmetic and main memory performance
        BRANCH         Branch prediction miss rate/ratio
     UOPS_EXEC         UOPs execution
           UPI         UPI data traffic
           TMA         Top down cycle allocation
        MEM_DP         Overview of arithmetic and main memory performance
     TLB_INSTR         L1 Instruction TLB miss rate/ratio
        DIVIDE         Divide unit information
       L2CACHE         L2 cache miss rate/ratio
```

```
           MEM         Main memory bandwidth in MBytes/s
      FLOPS_DP         Double Precision MFLOP/s
          DATA         Load to store ratio
   UOPS_RETIRE         UOPs retirement
      TLB_DATA         L2 data TLB miss rate/ratio
      FLOPS_SP         Single Precision MFLOP/s
  CYCLE_STALLS         Cycle Activities (Stalls)
    UOPS_ISSUE         UOPs issueing
         CLOCK         Power and Energy consumption
       L3CACHE         L3 cache miss rate/ratio
            L2         L2 cache bandwidth in MBytes/s
            L3         L3 cache bandwidth in MBytes/s
```

likwid on hydra II

```
hunold@hydra-head:~/tmp/papi/papi_mm$ likwid-perfctr -C S0:0 -g FLOPS_DP ./matrixmult_likwid
--------------------------------------------------------------------------------
CPU name:        Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz
CPU type:        Intel Skylake SP processor
CPU clock:        2.09 GHz
--------------------------------------------------------------------------------
Group 1: FLOPS_DP
+-----------------------------------------+---------+-------------+
|                  Event                  | Counter |  HWThread 0 |
+-----------------------------------------+---------+-------------+
|             INSTR_RETIRED_ANY           |  FIXC0  | 14789347909 |
|           CPU_CLK_UNHALTED_CORE         |  FIXC1  |  5458053255 |
|           CPU_CLK_UNHALTED_REF          |  FIXC2  |  3133374216 |
| FP_ARITH_INST_RETIRED_128B_PACKED_DOUBLE |   PMC0  |           0 |
|    FP_ARITH_INST_RETIRED_SCALAR_DOUBLE  |   PMC1  |   720000003 |
| FP_ARITH_INST_RETIRED_256B_PACKED_DOUBLE |   PMC2  |           0 |
| FP_ARITH_INST_RETIRED_512B_PACKED_DOUBLE |   PMC3  |           0 |
+-----------------------------------------+---------+-------------+
+----------------------+------------+
|        Metric        | HWThread 0 |
+----------------------+------------+
|  Runtime (RDTSC) [s] |     1.5109 |
| Runtime unhalted [s] |     2.6053 |
|      Clock [MHz]     |  3649.2955 |
|          CPI         |     0.3691 |
|     DP [MFLOP/s]     |   476.5248 |
|   AVX DP [MFLOP/s]   |          0 |
|  AVX512 DP [MFLOP/s] |          0 |
|   Packed [MUOPS/s]   |          0 |
|   Scalar [MUOPS/s]   |   476.5248 |
|  Vectorization ratio |          0 |
+----------------------+------------+
```

likwid on hydra III

- we see that the code does not use any vectorized instructions
- that is bad for performance
- so, let's turn vectorizations on
- also make sure that gcc does not optimize code away

```
hunold@hydra-head:~/tmp/papi/papi_mm$ gcc -march=native -O3  -o matrixmult_likwid matrixmult_likwid.c
```

likwid on hydra IV

```
hunold@hydra-head:~/tmp/papi/papi_mm$ likwid-perfctr -C S0:0 -g FLOPS_DP ./matrixmult_likwid
--------------------------------------------------------------------------------
CPU name:        Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz
CPU type:        Intel Skylake SP processor
--------------------------------------------------------------------------------
Group 1: FLOPS_DP
+-----------------------------------------+---------+------------+
|                  Event                  | Counter | HWThread 0 |
+-----------------------------------------+---------+------------+
|             INSTR_RETIRED_ANY           |  FIXC0  |  543125226 |
|           CPU_CLK_UNHALTED_CORE         |  FIXC1  |  711822567 |
|           CPU_CLK_UNHALTED_REF          |  FIXC2  |  419227956 |
| FP_ARITH_INST_RETIRED_128B_PACKED_DOUBLE |   PMC0  |          0 |
|    FP_ARITH_INST_RETIRED_SCALAR_DOUBLE  |   PMC1  |          3 |
| FP_ARITH_INST_RETIRED_256B_PACKED_DOUBLE |   PMC2  |  180000000 |
| FP_ARITH_INST_RETIRED_512B_PACKED_DOUBLE |   PMC3  |          0 |
+-----------------------------------------+---------+------------+

+----------------------+--------------+
|        Metric        |  HWThread 0  |
+----------------------+--------------+
|  Runtime (RDTSC) [s] |       0.2096 |
| Runtime unhalted [s] |       0.3398 |
|      Clock [MHz]     |    3557.1784 |
|          CPI         |       1.3106 |
|     DP [MFLOP/s]     |    3435.4493 |
|   AVX DP [MFLOP/s]   |    3435.4493 |
|  AVX512 DP [MFLOP/s] |            0 |
|   Packed [MUOPS/s]   |     858.8623 |
|   Scalar [MUOPS/s]   | 1.431437e-05 |
|  Vectorization ratio |     100.0000 |
+----------------------+--------------+
```

MPI - Profiling Support

MPI - Tool Support

- `http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf`, pages 585-
- MPI profiling interface
- requirements: MPI implementations must
  - "requires, in C and Fortran, an alternate entry point name, with the prefix `PMPI_` for each MPI function in each provided language binding and language support method"

MPI Profiler Implementation Example

```
static int totalBytes = 0;
static double totalTime = 0.0;
int MPI_Send(const void* buffer, int count, MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm)
{
  double tstart = MPI_Wtime();  /* Pass on all arguments */
  int extent;
  int result = PMPI_Send(buffer,count,datatype,dest,tag,comm);
  totalTime  += MPI_Wtime() - tstart;         /* and time          */
  MPI_Type_size(datatype, &extent);  /* Compute size */
  totalBytes += count*extent;
  return result;
}
```

source: http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

MPI - Profiling Interface

```
#ifdef PROFILELIB
#    ifdef __STDC__
#        define FUNCTION(name) P##name
#    else
#        define FUNCTION(name) P/**/name
#    endif
#else
#    define FUNCTION(name) name
#endif

/* ... */

int FUNCTION(MPI_Example)(/* appropriate args */)
{
    /* Useful content */
}
```

- compile then: `cc ... -lmyprof -lpmpi -lmpi`
- ## token pasting or token concatenation (https://gcc.gnu.org/onlinedocs/cpp/Concatenation.html)

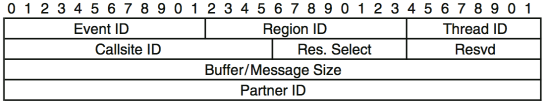
source: http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

IPM

IPM

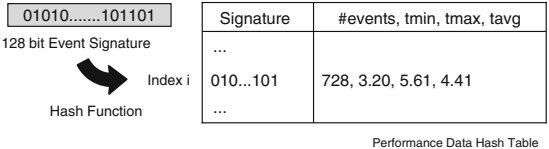
- IPM (Integrated Performance Monitoring) (LGPL)
  - `http://ipm-hpc.org/`
  - `https://github.com/nerscadmin/IPM`
  - originates from NERSC (LBNL)
- portable profiling infrastructure for parallel codes (mainly MPI)
- low-overhead profile and resource utilization in a parallel program

IPM



- event ID: 12 bits for event being monitored (e.g., completion of `MPI_Send`)
- region ID: events for regions marked by user
- thread ID: encodes ID of thread from which event originated
- callsite ID: derive call site of monitored function by walking call stack (e.g., `libunwind`)
- partner ID: communication partner (MPI operation) or file ID for file ops
  - for collective operations the peer rank is recorded
- buffer/message size: length of message (MPI) or nb. of bytes for I/O ops

IPM



source: [2]

- IPM observes events at runtime
- computes their signature and updates the events' statistics
- IPM uses a hash table to implement the performance data table
- event signature is used as the hash key
- hash value: number of occurrences, the minimum, maximum, and sum of the duration of the events

IPM internals - MPI_Send

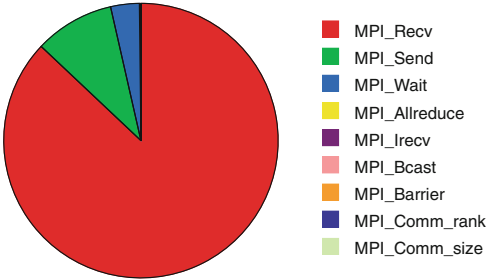
```c
int MPI_Send(void *sbuf, int scount,
MPI_Datatype stype, int dest, int tag,
MPI_Comm comm)
{
  int rv;
  int bytes, irank;
  int csite, idx, idx2, regid;
  double tstart, tstop, t;
  IPM_KEY_TYPE key;

  tstart = ipm_wtime();
  rv = PMPI_Send(sbuf,scount,stype,dest,tag,comm);
  tstop = ipm_wtime();
  if( ipm_state!=STATE_ACTIVE ) {
    return rv;
  }
  t=tstop-tstart;

  bytes=0; irank=0;
  IPM_MPI_BYTES_SCOUNT_C(bytes);
  IPM_MPI_RANK_DEST_C(irank);

  if( irank==MPI_ANY_SOURCE ) {
    irank = IPM_RANK_ANY_SOURCE;
  }
#ifdef HAVE_CALLPATH
  csite=get_callsite_id();
#else
  csite=0;
#endif
```

```c
  regid=ipm_rstackptr->id;

  /* check for out-of-range */
  if(irank<KEY_MIN_RANK || irank>KEY_MAX_RANK)
    irank=IPM_RANK_NULL;

  if(csite<KEY_MIN_CALLSITE || csite>KEY_MAX_CALLSITE)
    irank=IPM_CALLSITE_NULL;

  if(regid<KEY_MIN_REGION || regid>KEY_MAX_REGION)
    irank=IPM_REGION_NULL;

#ifdef HAVE_MPI_TRACE
  IPM_MPI_TRACE(MPI_SEND_ID_GLOBAL, "MPI_Send",
                irank, bytes, regid, csite);
#endif

  IPM_MPI_KEY(key, MPI_SEND_ID_GLOBAL, irank, bytes,
              regid, csite);

  IPM_HASH_HKEY(ipm_htable, key, idx);

/* removed some code */

  IPM_HASHTABLE_ADD(idx,t);

  return rv;
}
```

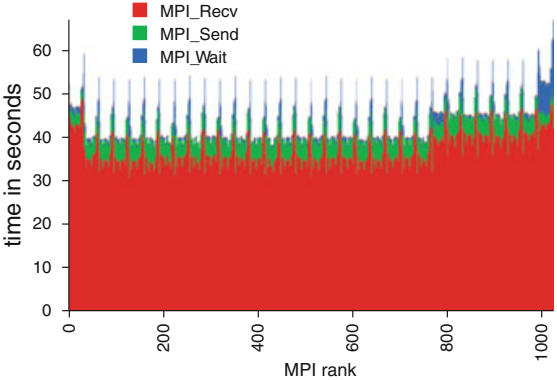
IPM [2]

```
##IPM##############################################################
#
# command   : ./a.out
# start     : Sun Mar 14 16:55:39 2010  host      : nid01829
# stop      : Sun Mar 14 17:04:33 2010  wallclock :   533.12
# mpi_tasks : 2048 on 1024 nodes        %comm     :    29.41
# omp_thrds : 6                         %omp      :    50.63
# files     : 12                        %i/o      :    12.09
# mem [GB]  : 2774.44                   gflop/sec :   418.58
#
#           :      [total]       <avg>        min        max
# wallclock :   1091671.57      533.04     532.99     533.12
# MPI       :    321034.43      156.76     109.03     239.23
# I/O       :    131947.08       64.43      11.83     113.87
# OMP       :    552665.28      269.86     205.07     305.36
# OMP idle  :     48262.98       23.57      21.30      27.40
# %wall     :
#   MPI     :                    29.41      20.45      44.88
#   OMP     :                    50.63      38.47      57.28
#   I/O     :                    12.09       2.22      21.36
# #calls    :
#   MPI     :     76235998       37224      37223      37320
# mem [GB]  :      2774.44        1.35       1.35       1.36
#
#                         [time]       [count]       <%wall>
# OMP_PARALLEL         552665.28     131439989         50.63
# MPI_Allreduce        247648.04      14438400         22.69
# fread                 69813.27       5488640          6.40
# ...
#
##################################################################
```

IPM [2]

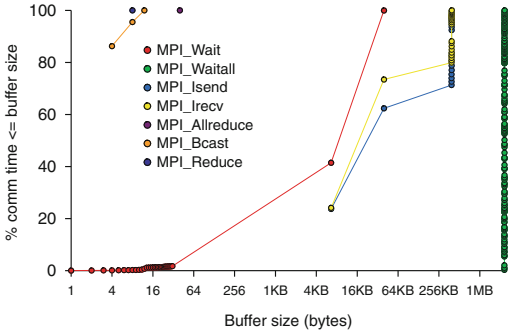


(a) MPI pie chart.

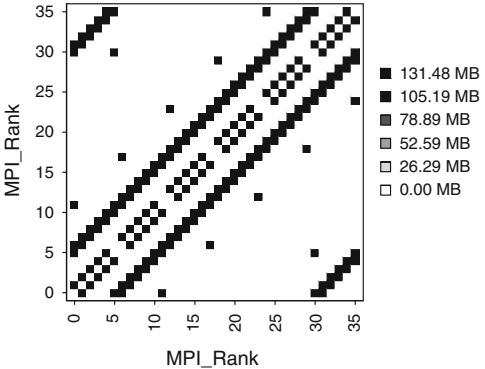


(b) Time in MPI routines by rank.

IPM [2]



(c) Cumulative distribution graph of buffer sizes.



(d) Communication topology graph.

mpiP

mpiP

- lightweight profiling library for MPI applications
  - `https://github.com/LLNL/mpiP`
- mpiP is a link-time library (similar to IPM)
- provides “call site” statistics, i.e., where was the MPI call issued

mpiP - example I

- let's use a very simply MPI application with 1 `MPI_Bcast` call

```c
#include "mpi.h"

int main(int argc, char **argv)
{
  int rank, value;

  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  value = rank;

  MPI_Bcast(&value, 1, MPI_INT, 0, MPI_COMM_WORLD);

  MPI_Finalize();

  return 0;
}
```

- load `mpip`

```
hunold@hydra-head:~/tmp/mpitools$ spack load mpip@3.5
```

mpiP - example II

- preloading library will do

```
hunold@hydra-head:~/tmp/mpitools$ LD_PRELOAD=/opt/spack/spack_git/opt/spack/linux-debian11-skylake_avx512/gcc-10.2.1/mpip-3.5-m3zfqt2bflfdfvssfnnt2addd7i3ty4y/lib/libmpiP.so srun -N 2 --ntasks-per-node=2 ./bcast
mpiP:
mpiP: mpiP V3.5.0 (Build Aug 10 2023/09:53:35)
mpiP:
mpiP:
mpiP: Storing mpiP output in [./bcast.4.1598966.1.mpiP].
mpiP:
```

mpiP - example III

```
hunold@hydra-head:~/tmp/mpitools$ cat bcast.4.1598966.1.mpiP
@ mpiP
@ Command : /home/hunold/tmp/mpitools/./bcast
@ Version                   : 3.5.0
@ MPIP Build date           : Aug 10 2023, 09:53:35
@ Start time                : 2023 12 28 09:18:20
@ Stop time                 : 2023 12 28 09:18:20
@ Timer Used                : PMPI_Wtime
@ MPIP env var              : [null]
@ Collector Rank            : 0
@ Collector PID             : 1598966
@ Final Output Dir          : .
@ Report generation         : Single collector task
@ MPI Task Assignment       : 0 hydra01
@ MPI Task Assignment       : 1 hydra01
@ MPI Task Assignment       : 2 hydra02
@ MPI Task Assignment       : 3 hydra02

---------------------------------------------------------------------------
@--- MPI Time (seconds) ---------------------------------------------------
---------------------------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0   0.000221   0.000203    91.90
   1   0.000708   0.000679    95.86
   2   0.000853   0.000837    98.11
   3   0.000277    0.00025    89.94
   *    0.00206    0.00197    95.57
---------------------------------------------------------------------------
@--- Callsites: 4 ---------------------------------------------------------
---------------------------------------------------------------------------
 ID Lev File/Address        Line Parent_Funct             MPI_Call
  1   0 0x7f6f62270b26           [unknown]                Bcast
```

mpiP - example IV

```
33   2   0 0x7f7f6cb53b26          [unknown]                Bcast
34   3   0 0x7f207a9b8b26          [unknown]                Bcast
35   4   0 0x7f857a5f8b26          [unknown]                Bcast
36 ---------------------------------------------------------------------------
37 @--- Aggregate Time (top twenty, descending, milliseconds) ----------------
38 ---------------------------------------------------------------------------
39 Call                 Site       Time    App%    MPI%       Count    COV
40 Bcast                   2      0.837   40.64   42.52           1   0.00
41 Bcast                   3      0.679   32.95   34.48           1   0.00
42 Bcast                   4       0.25   12.11   12.67           1   0.00
43 Bcast                   1      0.203    9.87   10.33           1   0.00
44 ---------------------------------------------------------------------------
45 @--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
46 ---------------------------------------------------------------------------
47 Call                 Site      Count      Total       Avrg  Sent%
48 Bcast                   4          1          4          4  25.00
49 Bcast                   3          1          4          4  25.00
50 Bcast                   2          1          4          4  25.00
51 Bcast                   1          1          4          4  25.00
52 ---------------------------------------------------------------------------
53 @--- Callsite Time statistics (all, milliseconds): 4 ----------------------
54 ---------------------------------------------------------------------------
55 Name              Site Rank  Count      Max     Mean      Min   App%   MPI%
56 Bcast                1    0      1    0.203    0.203    0.203  91.90 100.00
57 Bcast                1    *      1    0.203    0.203    0.203   9.87  10.33
58
59 Bcast                2    2      1    0.837    0.837    0.837  98.11 100.00
60 Bcast                2    *      1    0.837    0.837    0.837  40.64  42.52
61
62 Bcast                3    1      1    0.679    0.679    0.679  95.86 100.00
63 Bcast                3    *      1    0.679    0.679    0.679  32.95  34.48
64
65 Bcast                4    3      1     0.25     0.25     0.25  89.94 100.00
```

mpiP - example V

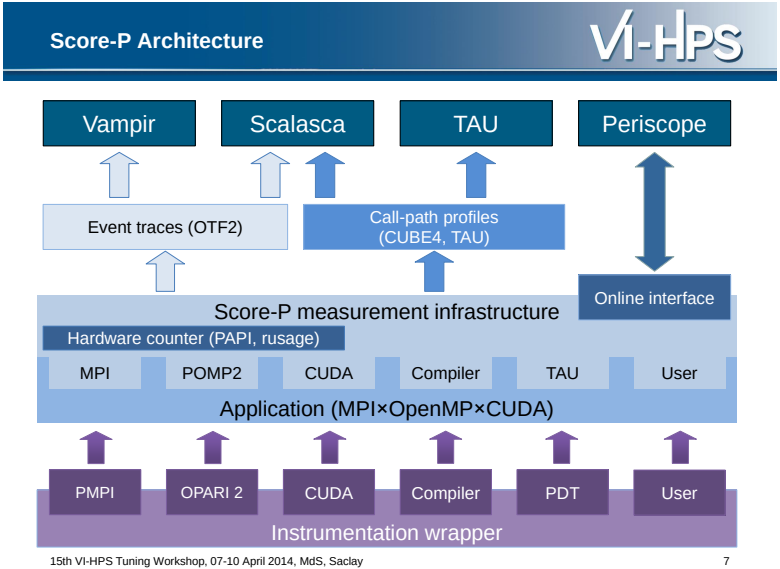
```
Bcast                 4    *      1     0.25     0.25     0.25  12.11  12.67
---------------------------------------------------------------------------
@--- Callsite Message Sent statistics (all, sent bytes) -------------------
---------------------------------------------------------------------------
Name              Site Rank   Count        Max       Mean        Min        Sum
Bcast                1    0       1          4          4          4          4
Bcast                1    *       1          4          4          4          4

Bcast                2    2       1          4          4          4          4
Bcast                2    *       1          4          4          4          4

Bcast                3    1       1          4          4          4          4
Bcast                3    *       1          4          4          4          4

Bcast                4    3       1          4          4          4          4
Bcast                4    *       1          4          4          4          4
---------------------------------------------------------------------------
@--- End of Report --------------------------------------------------------
---------------------------------------------------------------------------
```

Vampir

Vampir

- `http://www.vampir.eu`
- TU Dresden
- supports: MPI, OpenMP, Pthreads, CUDA, Java Threads
- license: commercial
- interactive trace visualization and analysis
- scalable to large trace data sizes (20 TByte)
- scalable to high parallelism (200000 processes)
- visualizes traces obtained from Score-P (New BSD Open Source license)
  - Scalable Performance Measurement Infrastructure for Parallel Codes
  - open-source tracing library
  - it works with Periscope, Scalasca, Vampir, and Tau

Score-P


Score-P Architecture
VI-HPS
Vampir
Scalasca
TAU
Periscope
Event traces (OTF2)
Call-path profiles
(CUBE4, TAU)
Online interface
Score-P measurement infrastructure
Hardware counter (PAPI, rusage)
MPI
POMP2
CUDA
Compiler
TAU
User
Application (MPI×OpenMP×CUDA)
PMPI
OPARI 2
CUDA
Compiler
PDT
User
Instrumentation wrapper
15th VI-HPS Tuning Workshop, 07-10 April 2014, MdS, Saclay
7

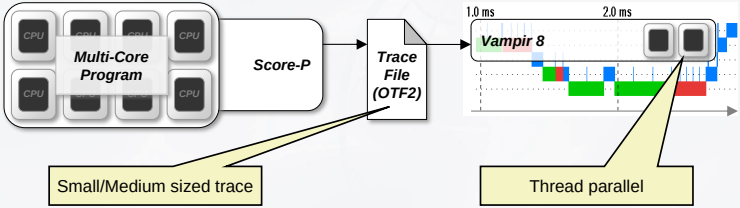
Slides on Vampir courtesy of Bert Wesarg and Ronny Tschüter (TU Dresden)

Vampir - Local Execution

Vampir – Visualization Modes (1)

- Directly on front end or local machine

```
% vampir
```



TECHNISCHE UNIVERSITÄT DRESDEN

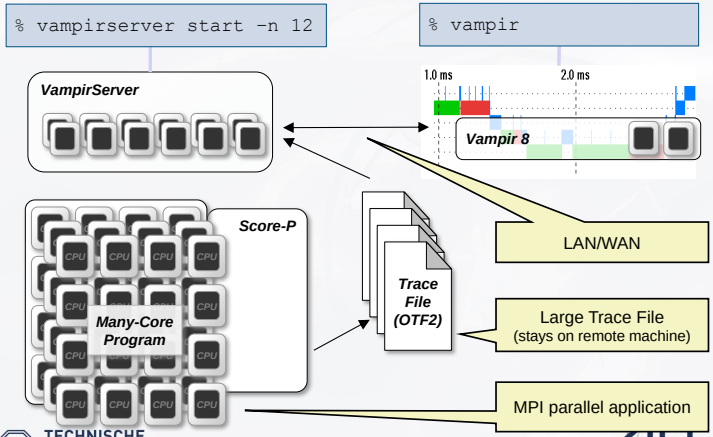
5

ZIH Center for Information Services & High Performance Computing

Vampir - Client/Server Architecture

Vampir – Visualization Modes (2)

- On local machine with remote VampirServer

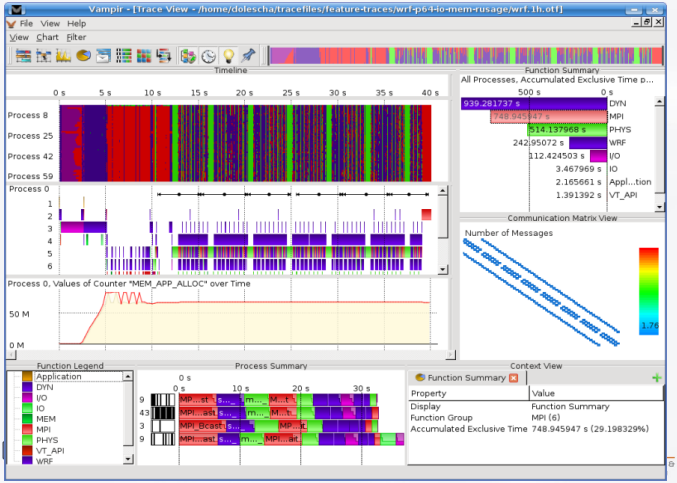


TECHNISCHE UNIVERSITÄT DRESDEN

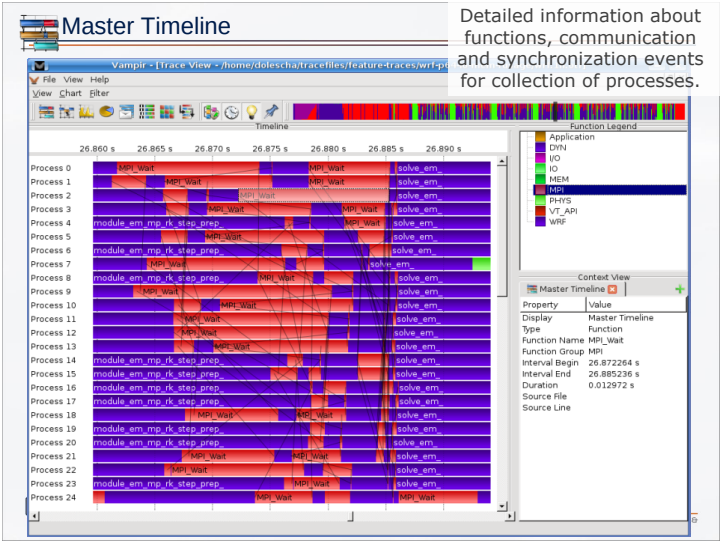
6

ZIH Center for Information Services & High Performance Computing

Vampir: Displays for a WRF Trace with 64 Processes

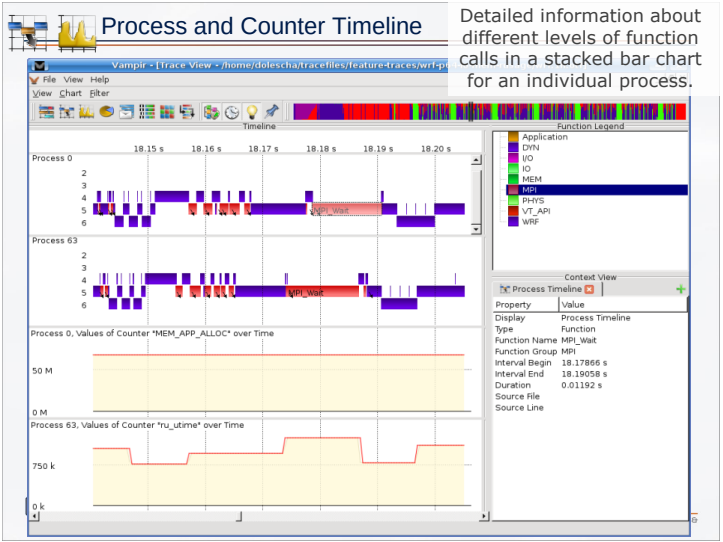


Vampir II



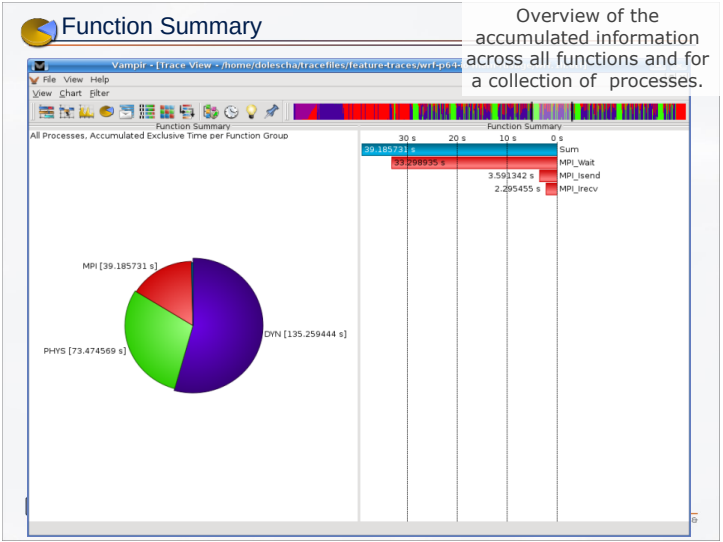
Detailed information about functions, communication and synchronization events for collection of processes.

Vampir III



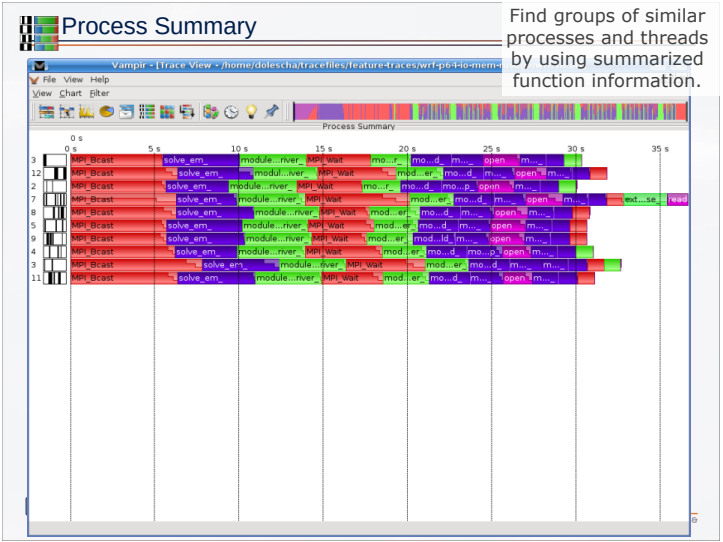
Detailed information about different levels of function calls in a stacked bar chart for an individual process.

Vampir IV



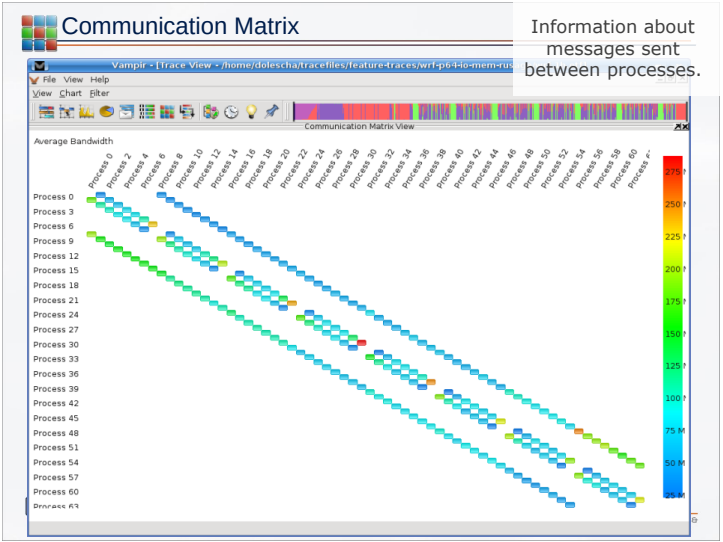
Overview of the accumulated information across all functions and for a collection of processes.

Vampir V



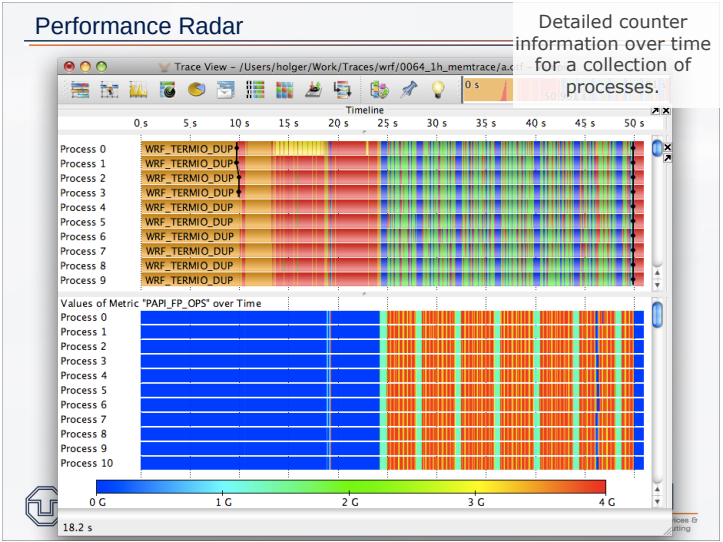
Find groups of similar processes and threads by using summarized function information.

Vampir VI



Information about messages sent between processes.

Vampir VII



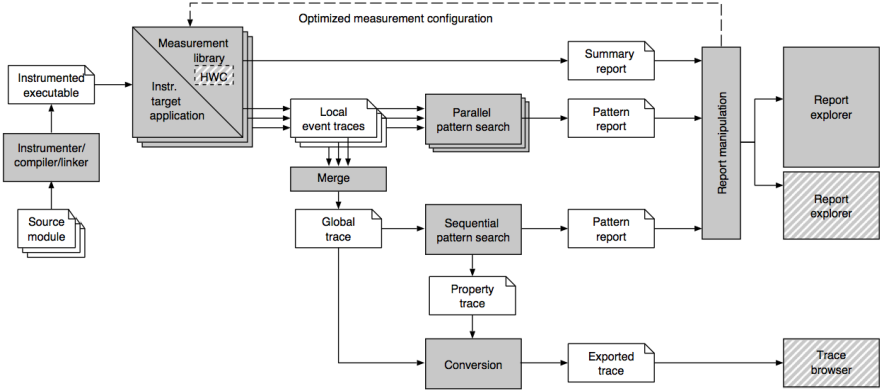
Detailed counter information over time for a collection of processes.

Scalasca

Scalasca

- `http://www.scalasca.org/`
- joint project of:
  - Jülich Supercomputing Centre
  - German Research School for Simulation Sciences (RWTH Aachen, JSC)
- Scalasca v2 is licensed under the BSD-style license
- performance toolset for analysis of application behavior on thousands of processors
- event tracing
- key feature
  - identify wait states in applications with very large numbers of processes
- supports measurement and analysis of the MPI, OpenMP
- `C`, `C++`, `Fortran`

Scalasca [3] I

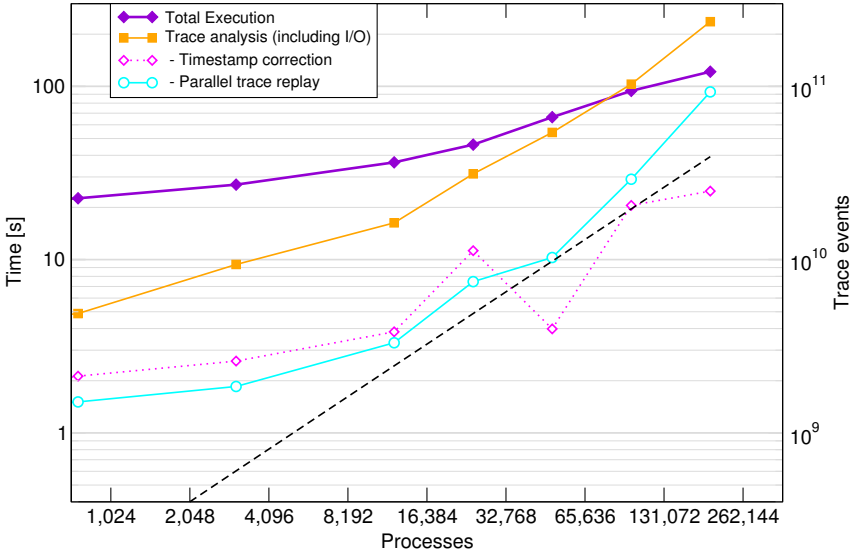


- instrumentation
  - MPI using PMPI profiling interface
  - OpenMP with POMP profiling interface
  - or custom instrumentation annotations in source code for important regions

Scalasca [3] II

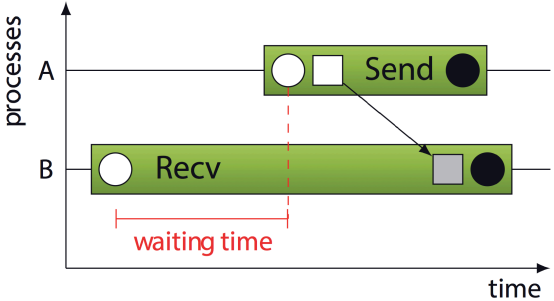
- provides a diagnostic method for localization of wait states by automatically searching event traces for characteristic patterns
- analyzes separate process-local trace files in parallel by replaying the original communication on as many cores as have been used to execute the target application itself
- traces
  - with globally synchronized high-resolution clocks (IBM BlueGene)
    - analyzed by comparing timestamps
  - other systems (Cray XT, clusters)
    - ability to synchronize inaccurate timestamps postmortem
    - problem of clock drifts, drifts varying in time
    - causes: violations of the logical event ordering
    - controlled logical clock algorithm [1]

Scalasca Performance



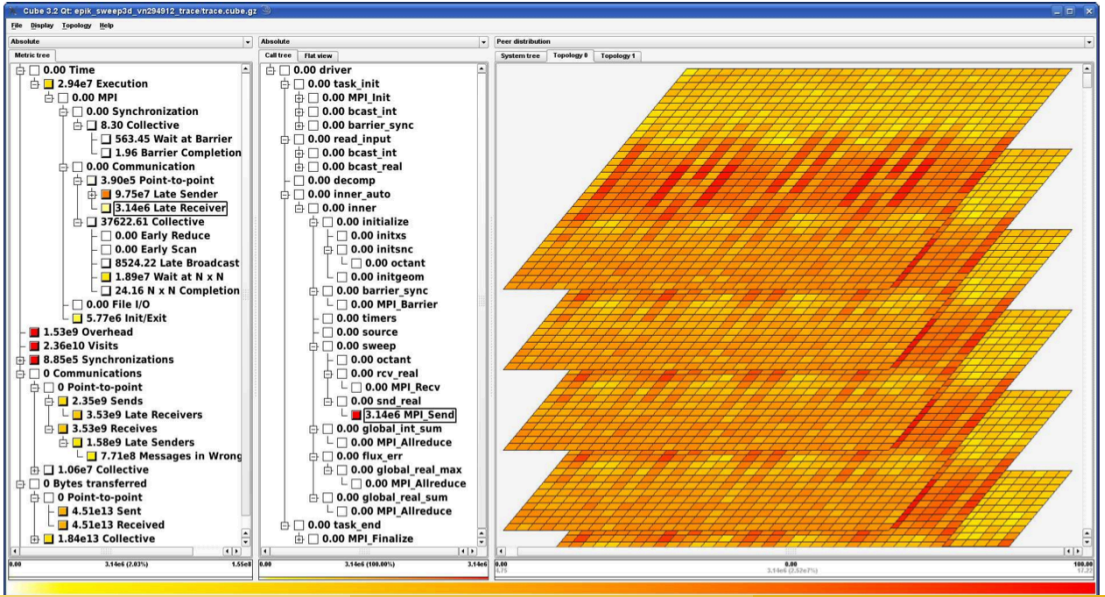
Scalasca - Example Pattern Late Sender

- late sender: blocking receiver waiting for message to arrive



source: http://www.fz-juelich.de/ias/jsc/EN/Expertise/Workshops/Exhibitions/ISC13/_flyer/Tools4-Scalasca_pdf.pdf?__blob=publicationFile

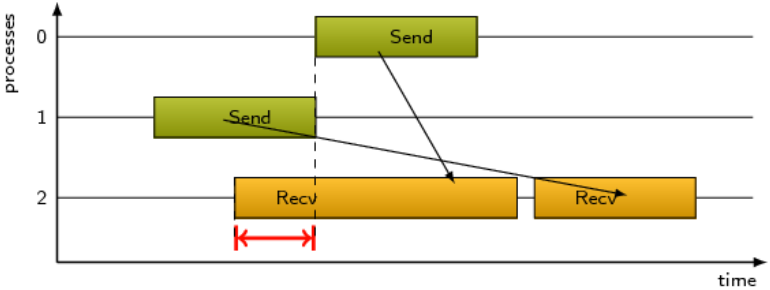
Scalasca - Late Sender/Receiver [3]



Scalasca - Late Sender/Receiver [3]

- 294912 processes, application: SWEEP3D, 72-rack Blue Gene/P
- distribution of the Late Receiver metric for the call path to the `MPI_Send`
- *Late Receiver waiting time* relatively small (2%),
- application spent roughly 28% (4.75s to 17.22s) in *Late Sender wait states*, where a receiver must wait for a message to arrive

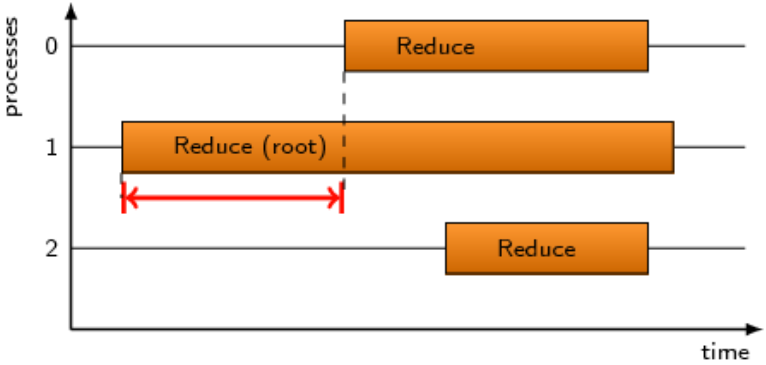
Scalasca - Patterns - Late Sender, Wrong Order Time / Different Sources



- analog: Late Sender, Wrong Order Time / Same Source

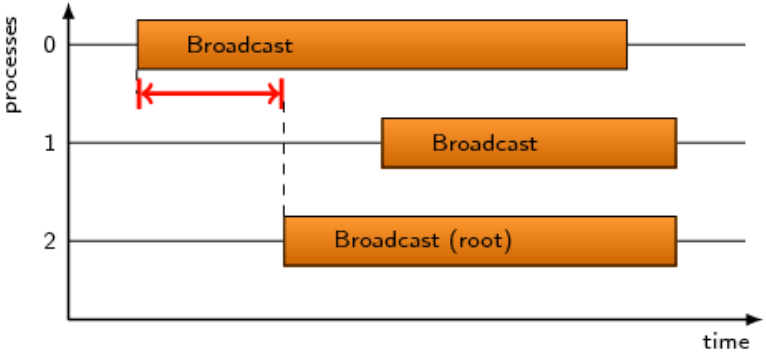
source: `http://apps.fz-juelich.de/scalasca/releases/scalasca/1.4/help/scalasca_patterns-1.4.html`

Scalasca - Patterns - Early Reduce



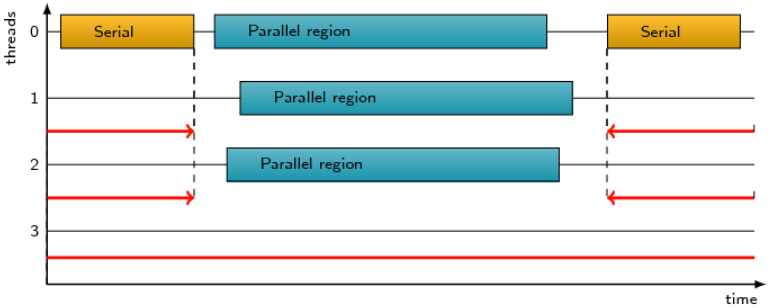
source: `http://apps.fz-juelich.de/scalasca/releases/scalasca/1.4/help/scalasca_patterns-1.4.html`

Scalasca - Patterns - Late Broadcast Time



source: `http://apps.fz-juelich.de/scalasca/releases/scalasca/1.4/help/scalasca_patterns-1.4.html`

Scalasca - Patterns - OpenMP Idle threads Time



source: `http://apps.fz-juelich.de/scalasca/releases/scalasca/1.4/help/scalasca_patterns-1.4.html`

Score-P - Practical Profiling I

- Score-P is a common measurement infrastructure for profiling and tracing of HPC applications
- now, we use it only for profiling MPI applications
- note: scorep has a dependency to an MPI library

```
hunold@hydra-head:~/tmp/mpitools$ spack find -l  scorep
-- linux-debian11-skylake_avx512 / gcc@10.2.1 -------------------
6bvrf65 scorep@7.1  kciilq7 scorep@8.0

-- linux-debian11-skylake_avx512 / gcc@12.1.0 -------------------
ensxoiw scorep@7.1
==> 3 installed packages

hunold@hydra-head:~/tmp/mpitools$ spack find -l -d scorep@8.0
-- linux-debian11-skylake_avx512 / gcc@10.2.1 -------------------
kciilq7 scorep@8.0
ll3mt2b     binutils@2.40
gn7zveh     cubelib@4.8
gm3nuiv     cubew@4.8
35yeg6c     opari2@2.0.6
xq7ylcd     openmpi@4.1.5
67hcjdc         hwloc@2.8.0
x5kxkil             libpciaccess@0.16
5yrb6yq                 util-macros@1.19.3
..

==> 1 installed package
```

Score-P - Practical Profiling II

```
hunold@hydra-head:~/tmp/mpitools$ spack find --loaded
-- linux-debian11-skylake_avx512 / gcc@10.2.1 -------------------
autoconf@2.69                       cubelib@4.8    libedit@3.1-20210216  ncurses@6.3        otf2@3.0        scorep@8.0          zstd@1.5.2
automake@1.16.5                     cubew@4.8      libiconv@1.16         numactl@2.0.14     papi@6.0.0.1    slurm@22.05.3
berkeley-db@18.1.40                 diffutils@3.8  libpciaccess@0.16     opa-psm2@11.2.230  perl@5.34.1     tar@1.34
binutils@2.40                       gdbm@1.19      libsigsegv@2.13       opari2@2.0.6       pigz@2.7        texinfo@6.5
bison@3.8.2                         gettext@0.21   libtool@2.4.7         openmpi@4.1.5      pkgconf@1.8.0   util-macros@1.19.3
bzip2@1.0.8                         hwloc@2.8.0    libxml2@2.10.1        openssh@9.0p1      pmix@4.2.1      xz@5.2.5
ca-certificates-mozilla@2022-07-19  krb5@1.19.3    m4@1.4.19             openssl@1.1.1q     readline@8.1.2  zlib@1.2.12
==> 43 loaded packages

hunold@hydra-head:~/tmp/mpitools$ which mpicc
/opt/spack/spack_git/opt/spack/linux-debian11-skylake_avx512/gcc-10.2.1/openmpi-4.1.5-xq7ylcdlsupyfzop5kuqxtensgbcoa2d/bin/mpicc
```

Score-P - Practical Profiling III

- compiling with `scorep` is very easy

```
hunold@hydra-head:~/tmp/mpitools$ scorep mpicc -o bcast bcast.c
hunold@hydra-head:~/tmp/mpitools$ ldd ./bcast
        linux-vdso.so.1 (0x00007fff0039a000)
        libscorep_adapter_compiler_event.so.11 =>
       ↪  /opt/spack/spack_git/opt/spack/linux-debian11-skylake_avx512/gcc-10.2.1/scorep-8.0-kciilq7zls2daoqjpkpzcddiyaabvn4c/lib/libscorep_adapter_compiler_ev
       ↪  (0x00007f186317e000)
        libscorep_adapter_mpi_event.so.0 =>
       ↪  /opt/spack/spack_git/opt/spack/linux-debian11-skylake_avx512/gcc-10.2.1/scorep-8.0-kciilq7zls2daoqjpkpzcddiyaabvn4c/lib/libscorep_adapter_mpi_event.s
       ↪  (0x00007f18630dd000)
        libscorep_measurement.so.11 =>
       ↪  /opt/spack/spack_git/opt/spack/linux-debian11-skylake_avx512/gcc-10.2.1/scorep-8.0-kciilq7zls2daoqjpkpzcddiyaabvn4c/lib/libscorep_measurement.so.11
       ↪  (0x00007f186301d000)
        libscorep_adapter_compiler_mgmt.so.11 =>
       ↪  /opt/spack/spack_git/opt/spack/linux-debian11-skylake_avx512/gcc-10.2.1/scorep-8.0-kciilq7zls2daoqjpkpzcddiyaabvn4c/lib/libscorep_adapter_compiler_mg
       ↪  (0x00007f186300a000)
        libscorep_adapter_mpi_mgmt.so.0 =>
       ↪  /opt/spack/spack_git/opt/spack/linux-debian11-skylake_avx512/gcc-10.2.1/scorep-8.0-kciilq7zls2daoqjpkpzcddiyaabvn4c/lib/libscorep_adapter_mpi_mgmt.so
       ↪  (0x00007f1862fdb000)
        ...
```

Score-P - Practical Profiling IV

- now, simply run the MPI application, which create a scorep directory

```
hunold@hydra-head:~/tmp/mpitools$ srun -N 2 --ntasks-per-node=2 ./bcast
hunold@hydra-head:~/tmp/mpitools$ ls scorep-20231230_1628_29731304861709958/
MANIFEST.md  profile.cubex  scorep.cfg
```

- the cubex file contains the profile

```
hunold@hydra-head:~/tmp/mpitools$ scorep-score -r ./scorep-20231230_1628_29731304861709958/profile.cubex

Estimated aggregate size of event trace:                   813 bytes
Estimated requirements for largest trace buffer (max_buf): 204 bytes
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       4097kB
(hint: When tracing set SCOREP_TOTAL_MEMORY=4097kB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)

flt     type max_buf[B] visits time[s] time[%] time/visit[us]  region
         ALL        203     24    0.30   100.0       12523.32  ALL
         MPI        138     16    0.29    97.9       18388.87  MPI
      SCOREP         41      4    0.01     2.1        1553.01  SCOREP
         COM         24      4    0.00     0.0          31.46  COM

         MPI         66      4    0.00     0.1          71.02  MPI_Bcast
      SCOREP         41      4    0.01     2.1        1553.01  bcast
         MPI         24      4    0.00     0.0           4.90  MPI_Comm_rank
         MPI         24      4    0.01     2.0        1474.35  MPI_Finalize
         MPI         24      4    0.29    95.8       72005.20  MPI_Init
         COM         24      4    0.00     0.0          31.46  main
```

Score-P - Practical Profiling V

- let's make it more interesting and increase the number of bcasts

```
hunold@hydra-head:~/tmp/mpitools$ cat bcast.c |grep -B 1 MPI_Bcast
  for(int i=0; i<2000; i++)
    MPI_Bcast(&value, 1, MPI_INT, 0, MPI_COMM_WORLD);

hunold@hydra-head:~/tmp/mpitools$ scorep mpicc -o bcast bcast.c
hunold@hydra-head:~/tmp/mpitools$ srun -N 2 --ntasks-per-node=2 ./bcast
hunold@hydra-head:~/tmp/mpitools$ scorep-score -r ./scorep-20231230_16
scorep-20231230_1628_29731304861709958/ scorep-20231230_1635_29732235207011280/
hunold@hydra-head:~/tmp/mpitools$ scorep-score -r ./scorep-20231230_1635_29732235207011280/profile.cubex

Estimated aggregate size of event trace:                   517kB
Estimated requirements for largest trace buffer (max_buf): 130kB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       4097kB
(hint: When tracing set SCOREP_TOTAL_MEMORY=4097kB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)

flt     type max_buf[B] visits time[s] time[%] time/visit[us]  region
         ALL    132,137  8,020    0.40   100.0          49.51  ALL
         MPI    132,072  8,012    0.39    99.3          49.20  MPI
      SCOREP         41      4    0.00     0.1         136.25  SCOREP
         COM         24      4    0.00     0.6         575.08  COM

         MPI    132,000  8,000    0.01     2.6           1.30  MPI_Bcast
      SCOREP         41      4    0.00     0.1         136.25  bcast
         MPI         24      4    0.00     0.0           6.10  MPI_Comm_rank
         MPI         24      4    0.00     0.0          36.03  MPI_Finalize
         MPI         24      4    0.38    96.6       95898.37  MPI_Init
         COM         24      4    0.00     0.6         575.08  main
```

Score-P - Practical Profiling VI

- consider setting

```
export SCOREP_TIMER=clock_gettime
```

since `Score-P` uses TSC registers by default, which may be inaccurate on `hydra` (no, it will be inaccurate)

HPCToolkit I

- HPCToolkit (Rice University)
- `http://hpctoolkit.org/`
- statistical sampling of timers and hardware performance counters
- tries to avoid blind spot (using sampling)
- call path profiling
  - associate costs with context
- uses call stack unwinding
- sampling: measurement has low overhead and scales to large parallel system

HPCToolkit II

- you will need an X11 environment, or inspect the trace later on your laptop

```
ssh -X hydrax
Linux hydra-head 5.10.0-18-amd64 #1 SMP Debian 5.10.140-1 (2022-09-02) x86_64
##############################################
@HPC students: Please do not run large jobs using
'mpirun' on hydra-head.
For benchmark runs, use 'srun'.
##############################################
Last login: Sat Dec 30 17:01:47 2023 from 128.131.171.7
hunold@hydra-head:~$ xeyes

hunold@hydra-head:~/tmp/mpitools$ spack load hpctoolkit@2022.05.15
hunold@hydra-head:~/tmp/mpitools$ which mpicc
/opt/spack/spack_git/opt/spack/linux-debian11-skylake_avx512/gcc-10.2.1/openmpi-4.1.4-6l5aunxh4ktbpsxrbglqvcqedqdqgkud/bin/mpicc
```

HPCToolkit III

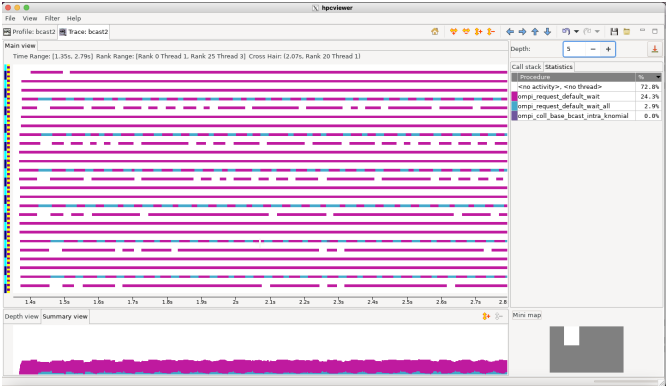
- for this test, we broadcast a little more data to make the program running longer

```c
int n  = 10000000;

/* ... */

msg = (double*)malloc(n*sizeof(double));

for(int i=0; i<100; i++)
  MPI_Bcast(msg, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);

free(msg);
```

```
hunold@hydra-head:~/tmp/mpitools$ mpicc -o bcast2 bcast2.c
hunold@hydra-head:~/tmp/mpitools$ srun -N 32 --ntasks-per-node=2 hpcrun -t ./bcast2
hunold@hydra-head:~/tmp/mpitools$ hpcprof hpctoolkit-bcast2-measurements-4362799
msg: Line map : /home/hunold/tmp/mpitools/bcast2
msg: Line map : /usr/lib/x86_64-linux-gnu/ld-2.31.so
...
msg: Populating Experiment database: /home/hunold/tmp/mpitools/hpctoolkit-bcast2-database-4362799
```

HPCToolkit IV

```
hunold@hydra-head:~/tmp/mpitools$ hpcviewer hpctoolkit-bcast2-database-4362799
```



References I

[1] D. Becker, R. Rabenseifner, and F. Wolf. "Timestamp Synchronization for Event Traces of Large-scale Message-passing Applications". In: *Proceedings of the 14th European Conference on Recent Advances in Parallel Virtual Machine and Message Passing Interface*. PVM/MPI'07. Paris, France: Springer-Verlag, 2007, pp. 315–325. ISBN: 3-540-75415-6, 978-3-540-75415-2.

[2] K. Fürlinger, N. J. Wright, D. Skinner, C. Klausecker, and D. Kranzlmüller. "Effective Holistic Performance Measurement at Petascale Using IPM". English. In: *Competence in High Performance Computing 2010*. Ed. by C. Bischof, H.-G. Hegering, W. E. Nagel, and G. Wittum. Springer Berlin Heidelberg, 2012, pp. 15–26. ISBN: 978-3-642-24024-9. DOI: 10.1007/978-3-642-24025-6_2.

[3] M. Geimer, F. Wolf, B. J. N. Wylie, E. Ábrahám, D. Becker, and B. Mohr. "The Scalasca performance toolset architecture". In: *Concurrency and Computation: Practice and Experience* 22.6 (2010), pp. 702–719.

[4] S. L. Graham, P. B. Kessler, and M. K. Mckusick. "Gprof: A Call Graph Execution Profiler". In: *Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction*. SIGPLAN '82. New York, NY, USA: ACM, 1982, pp. 120–126. ISBN: 0-89791-074-5. DOI: 10.1145/800230.806987.

References II

[5] S. Hunold and T. Rauber. "Automatic Tuning of PDGEMM Towards Optimal Performance". In: *Euro-Par*. Ed. by J. C. Cunha and P. D. Medeiros. Vol. 3648. Lecture Notes in Computer Science. Springer, 2005, pp. 837–846. ISBN: 3-540-28700-0.