Basic Profiling Tools and Hardware Counter Usage

Get started with some standard tools on blizzard

Hendryk Bockelmann, DKRZ

DKRZ
DEUTSCHES
KLIMARECHENZENTRUM

Access to performance analysis tools

- standard profiling available (like `-p` and `-pg` compiler flags)
- source `/usr/lpp/ppe.hpct/env_sh` for IBM tools (libmpitrace and HPM) to set `IHPCT_BASE` and paths
- use modules for vampirtrace, scalasca and vampir(server)

module environment:

- `module av`: shows all modules available
- `module add/rm`: adds, removes modules
- on blizzard use: `scalasca/1.4.2rc3-aixpoe-ibm`, `vampirtrace/5.13rc1-aixpoe-ibm` (latest versions), maybe go back to stable versions ...
- on lizard use: `vampir/vampirclient-7.5.0`

Sequential profiling with prof

- displays object file profile data
- for each text symbol in object file the percentage of execution time, number of times that function was called, and the average number of milliseconds per call is shown
- easy to use: compile and link with additional `-p` option, run your program as usual
- produce profiling information:

```
prof ./foo.x -m mon.out > foo.mon.rpt
```

- output is flat profile of called functions, even MPI calls are profiled at lowest level (ie. `_lapi_*`)

Seq profiling with prof (example)

File Edit View Search Terminal Help

Name	%Time	Seconds	Cumsecs	#Calls	msec/call
._lapi_shm_dispatche	18.1	37.20	37.20		
._lapi_dispatcher(un	9.8	20.10	57.30		
.LAPI__Msgpoll	8.6	17.61	74.91		
.__mo_diffusion_NMOD	5.6	11.55	86.46		
.__mo_adpo_NMOD_ocad	5.3	10.97	97.43		
.__mo_adpo_NMOD_ocad	5.0	10.1	107.62		
.__mo_baroclinic_NMO	3.6	7.50	115.12		
.__mo_iolist_NMOD_av	3.2	6.58	121.70		
.__mo_adpo_NMOD_ocad	3.1	6.44	128.14		
._is_yield_queue_emp	3.0	6.20	134.34		
.__mo_ocjitr_NMOD_oc	2.7	5.56	139.90		
.__mo_boundsexch_NMO	2.4	4.98	144.88	18939	0.2629
.__mo_ocmodmom_NMOD_	2.2	4.53	149.41		
.__mo_adisit_NMOD_po	1.9	3.99	153.40		
.do_ccl	1.7	3.55	156.95		
.__mo_diagnosis_NMOD	1.4	2.91	159.86		
._lapi_memcpy(void(*	1.0	2.07	161.93		
.__mo_adpo_NMOD_adve	1.0	2.00	163.93		
.__mo_ocvisc_NMOD_oc	1.0	1.99	165.92		
.__mcount	0.9	1.80	167.72		
.__mo_iolist_NMOD_fi	0.9	1.78	169.50		
.__mo_ocean_vertical	0.8	1.62	171.12		
.__mo_ocjitr_NMOD_oc	0.8	1.62	172.74		
.ocuad_	0.7	1.39	174.13		
.__mo_commo1_NMOD_lw	0.7	1.35	175.4897275384		0.0000
.__mo_adpo_NMOD_ocad	0.7	1.34	176.82		

:

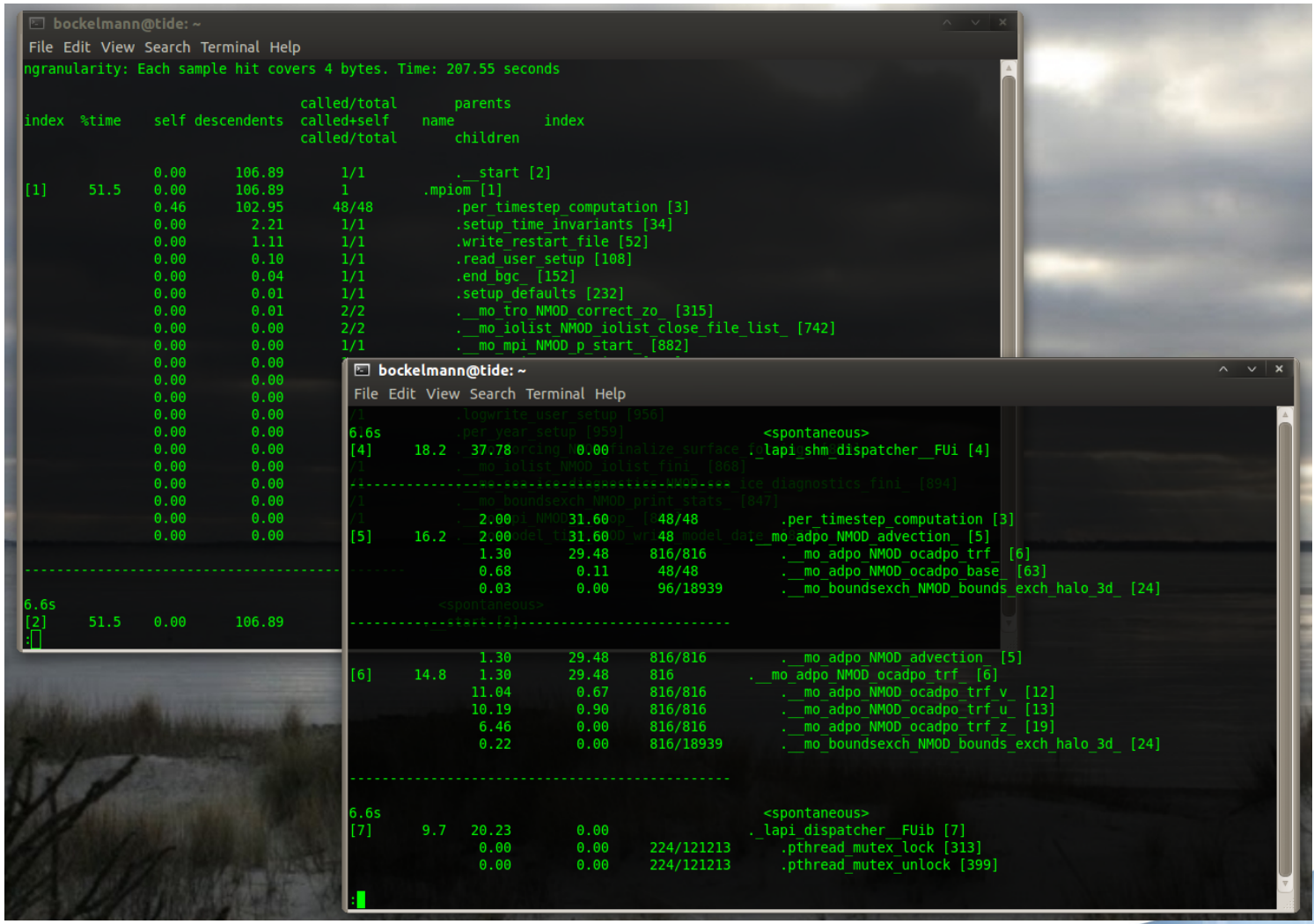
Sequential profiling with gprof

- in addition to prof, gprof shows the call-graph data and not only flat profiles
- easy to use, available on most systems: compile and link with additional `-pg` option, run your program as usual
- produce profiling information:

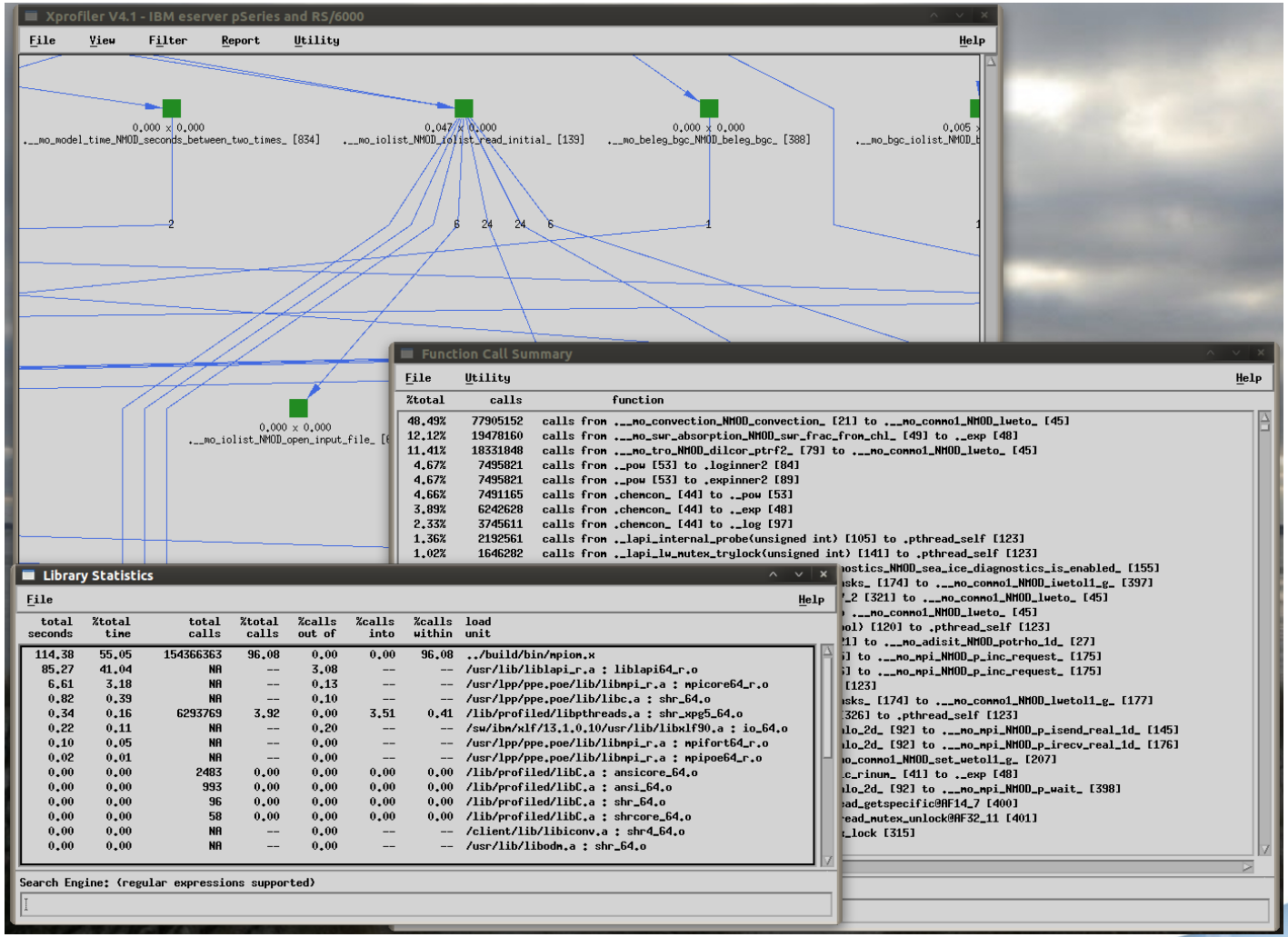
  ```
  gprof ./foo.x gmon.out > foo.gmon.rpt
  ```

- drawback: profiling support is added by the compiler, so if you wish to obtain profiling information from any shared libraries, you need to also compile them with `-pg` (therefore MPI calls `_lapi_*` appear spontanously)

Seq profiling with gprof (example)



Use xprofiler for gmon.out



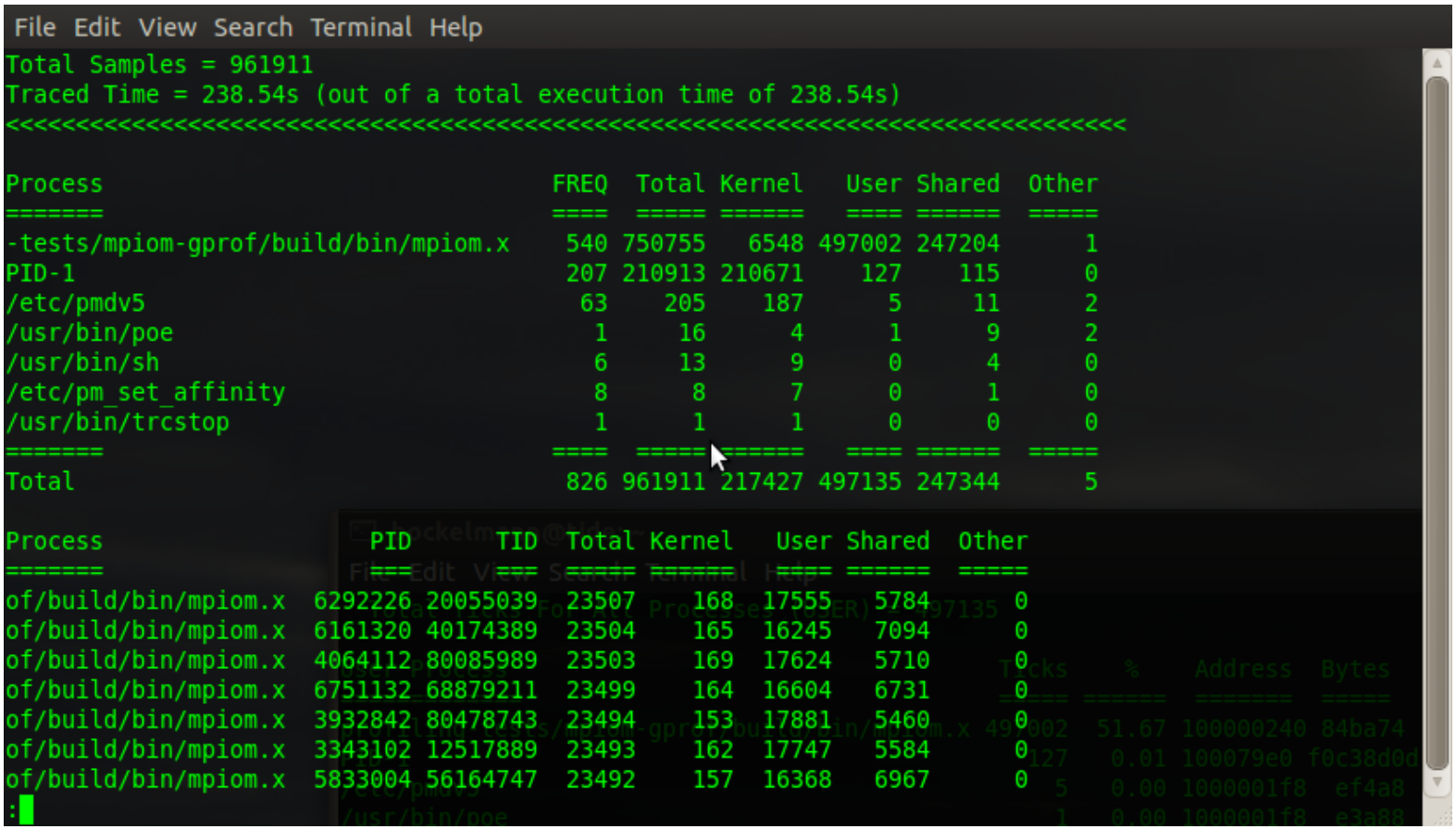
Processor usage report with tprof

- tprof charges processor time to object files, processes, threads, subroutines (user mode, kernel mode and shared library) - using AIX trace utility started in background
- advantage: subroutine-level profiling without modifying executable programs (no recompile, relink needed)
- modify batch script:

```
trcstop 2>/dev/null
tprof -usz -p <binary> -x poe /path/to/binary
```

- profiles all processes on system (use node `not_shared`)
- output of tprof is very flat, but shows all needed information
- only one node will be profiled (the one on which poe is started!)

System profiling with tprof (example)



```
File  Edit  View  Search  Terminal  Help
Total Samples = 961911
Traced Time = 238.54s (out of a total execution time of 238.54s)
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

Process                                 FREQ  Total Kernel   User Shared  Other
=======                                 ====  ===== ======   ==== ======  =====
-tests/mpiom-gprof/build/bin/mpiom.x     540 750755   6548 497002 247204      1
PID-1                                    207 210913 210671    127    115      0
/etc/pmdv5                                63    205    187      5     11      2
/usr/bin/poe                               1     16      4      1      9      2
/usr/bin/sh                                6     13      9      0      4      0
/etc/pm_set_affinity                       8      8      7      0      1      0
/usr/bin/trcstop                           1      1      1      0      0      0
=======                                 ====  ===== ======   ==== ======  =====
Total                                    826 961911 217427 497135 247344      5

Process                  PID      TID Total Kernel   User Shared  Other
=======                  ===      === ===== ======   ==== ======  =====
of/build/bin/mpiom.x  6292226 20055039 23507    168  17555   5784      0
of/build/bin/mpiom.x  6161320 40174389 23504    165  16245   7094      0
of/build/bin/mpiom.x  4064112 80085989 23503    169  17624   5710      0
of/build/bin/mpiom.x  6751132 68879211 23499    164  16604   6731      0
of/build/bin/mpiom.x  3932842 80478743 23494    153  17881   5460      0
of/build/bin/mpiom.x  3343102 12517889 23493    162  17747   5584      0
of/build/bin/mpiom.x  5833004 56164747 23492    157  16368   6967      0
:
```

System profiling with tprof (example)

```
File  Edit  View  Search  Terminal  Help
 Total Ticks For All Processes (USER) = 497135

User Process                                  Ticks    %    Address  Bytes
============                                  ===== ======  =======  =====
profiling-tests/mpiom-gprof/build/bin/mpiom.x 497002  51.67 100000240 84ba74
PID-1                                           127   0.01 100079e0 f0c38d0d
/etc/pmdv5                                        5   0.00 1000001f8  ef4a8
/usr/bin/poe                                      1   0.00 1000001f8  e3a88

  Profile: /work/k20200/k202082/profiling-tests/mpiom-gprof/build/bin/mpiom.x

  Total Ticks For All Processes (/work/k20200/k202082/profiling-tests/mpiom-gprof/build/bin/mpiom.x) = 497002

Subroutine                Ticks    %   Source                Address  Bytes
==========                ===== ====== ======                =======  =====
ffusion_NMOD_octdiff_trf_ 86799   9.02 src/mo_diffusion.f90   623c00   d3a0
o_adpo_NMOD_ocadpo_trf_v_ 45991   4.78 n/../src/mo_adpo.f90   67f760   3780
o_adpo_NMOD_ocadpo_trf_u_ 43318   4.50 n/../src/mo_adpo.f90   682ae0   3f20
o_adpo_NMOD_ocadpo_trf_z_ 36246   3.77 n/../src/mo_adpo.f90   686e00   4940
o_baroclinic_NMOD_occlit_ 24764   2.57 rc/mo_baroclinic.f90   64ad60  110a0
glist_element_accumulate_ 20318   2.11 ../src/mo_iolist.f90   56fde0   4700
o_ocjitr_NMOD_ocjitr_trf_ 19923   2.07 ../src/mo_ocjitr.f90   69fae0   5940
.carchm_                  18691   1.94 rc_hamocc/carchm.f90   713040   4900
NMOD_bounds_exch_halo_3d_ 15508   1.61 rc/mo_boundsexch.f90   116780   9620
o_ocmodmom_NMOD_ocmodmom_ 14818   1.54 /src/mo_ocmodmom.f90   65be60   d400
.ocprod_                  13556   1.41 rc_hamocc/ocprod.f90   6ff240   fd80
mo_adisit_NMOD_potrho_1d_ 13028   1.35 ../src/mo_adisit.f90   5de5a0    760
fusion_NMOD_octdiff_base_ 12944   1.35 src/mo_diffusion.f90   61c720   74e0
:
```

System profiling with tprof (example)

```
File  Edit  View  Search  Terminal  Help

  Total Ticks For All Processes (SH-LIBs) = 247344

Shared Object                                      Ticks    %    Address  Bytes
=============                                      ===== ======  =======  =====
/usr/lib/liblapi_r.a[liblapi64_r.o]               218930  22.76 900000002380480 15f734
usr/lpp/ppe.poe/lib/libmpi_r.a[mpicore64_r.o]      25859   2.69 9000000020a2c00 27d15c
/usr/lpp/ppe.poe/lib/libc.a[shr_64.o]               1666   0.17 900000000000900 3108fc
/usr/lib/libpthreads.a[shr_xpg5_64.o]                440   0.05 9000000005e0200  30a90
usr/lpp/ppe.poe/lib/libmpi_r.a[mpifort64_r.o]        198   0.02 900000001b17fc0   fe24
/usr/lpp/ppe.poe/lib/libmpi_r.a[mpipoe64_r.o]         70   0.01 90000000206f280  271b7
/usr/lib/libtrace.a[shr.o]                            47   0.00 d1d5e280  2e17c
ibm/xlf/13.1.0.10/usr/lib/libxlf90.a[io_64.o]         28   0.00 90000000504cf00  67be0
/usr/lib/libc.a[shr_64.o]                             23   0.00 900000000000900 3108fc
/usr/lib/libpthreads.a[shr_xpg5.o]                    21   0.00 d04ea180  3157b
/usr/lib/libc.a[shr.o]                                19   0.00 d0118680 31404c
/usr/lib/libptools.a[shr.o]                           11   0.00 d42b2280  33228
/usr/lib/libs.a[shr_64.o]                              7   0.00 900000000000900 3108fc
/usr/lib/IbBaseLib.a[shr_64.o]                         5   0.00 9000000007f6c00  24740
/usr/lib/libodm.a[shr_64.o]                            5   0.00 9000000003ebe80  15e48
/usr/lib/libllrapi.a[shr.o]                            5   0.00 d2e09260 6d9dfc
/usr/lib/libhal_ib.a[hal_ib64.o]                       3   0.00 90000000282d600   ef20
/usr/lpp/ppe.poe/lib/libc.a[shr.o]                     2   0.00 d0118680 31404c
/usr/lib/libllrapi.a[shr_64.o]                         2   0.00 900000000a51ce0 6e58f0
/usr/lib/libpnsd.a[shr_64.o]                           1   0.00 90000000274cd80  c8030
/opt/freeware/lib/ganglia/modcpu.so                    1   0.00 d05c6128   4b0f
/usr/lib/libC.a[ansicore_64.o]                         1   0.00 9000000005a3400   f850

:
```

Summary of *prof

src change, output given:

- gprof: standard, supported profiling tool on many UNIX systems - no need to learn new stuff
- tprof: collects data with no impact on execution time, works on optimized binaries without any need for recompilation - prof, gprof might have overhead, might not work on optimized binary, need recompile, relink
- prof, gprof: provide subprogram profiling, exact counts of number of times every subprogram is called - tprof does not
- gprof: provides call graph - prof, tprof do not

Summary of *prof

what data is given:

- all *prof obtain processor consumption estimates for each subprogram by sampling program counter of user program
- tprof collects processor usage information for whole system - prof, gprof get profiling information for single program & time in user mode

treatment of MPI data:

- all *prof tools give flat information on CPU-time used
- all MPI-stuff is hidden, each process gives its own profile (no correlations can be drawn)

$\rightarrow$ MPI profiling needed

MPI profiling

What we know by now:

- time spent in user subroutines
- time spent in MPI-lib

What we do not know by now:

- which MPI routines take the time
- correlations between MPI-tasks (send/receive/wait)

$\rightarrow$ tools are needed (this is why you are here)

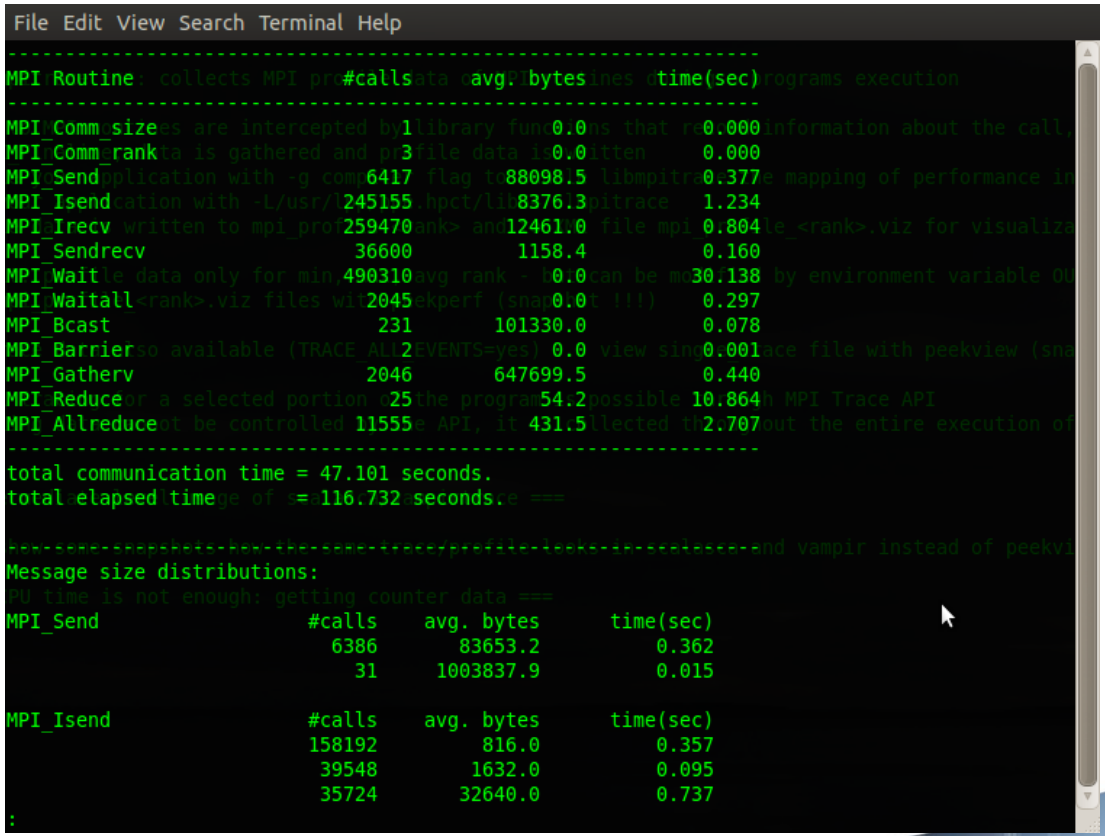
- libmpitrace
- scalasca
- vampir(trace)

HPCTlib: libmpitrace

- calls to MPI routines are intercepted by library functions
- MPI profile data of MPI routines can be collected during a programs execution
- on `MPI_Finalize`, data is gathered and profile data is written
- compile your application with `-g` to enable mapping of performance information back to the application source
- link your application with `-L/usr/lpp/ppe.hpct/lib64 -lmpitrace`
- profile data written to `mpi_profile.<rank>` and XML file `mpi_profile_<rank>.viz` for visualization

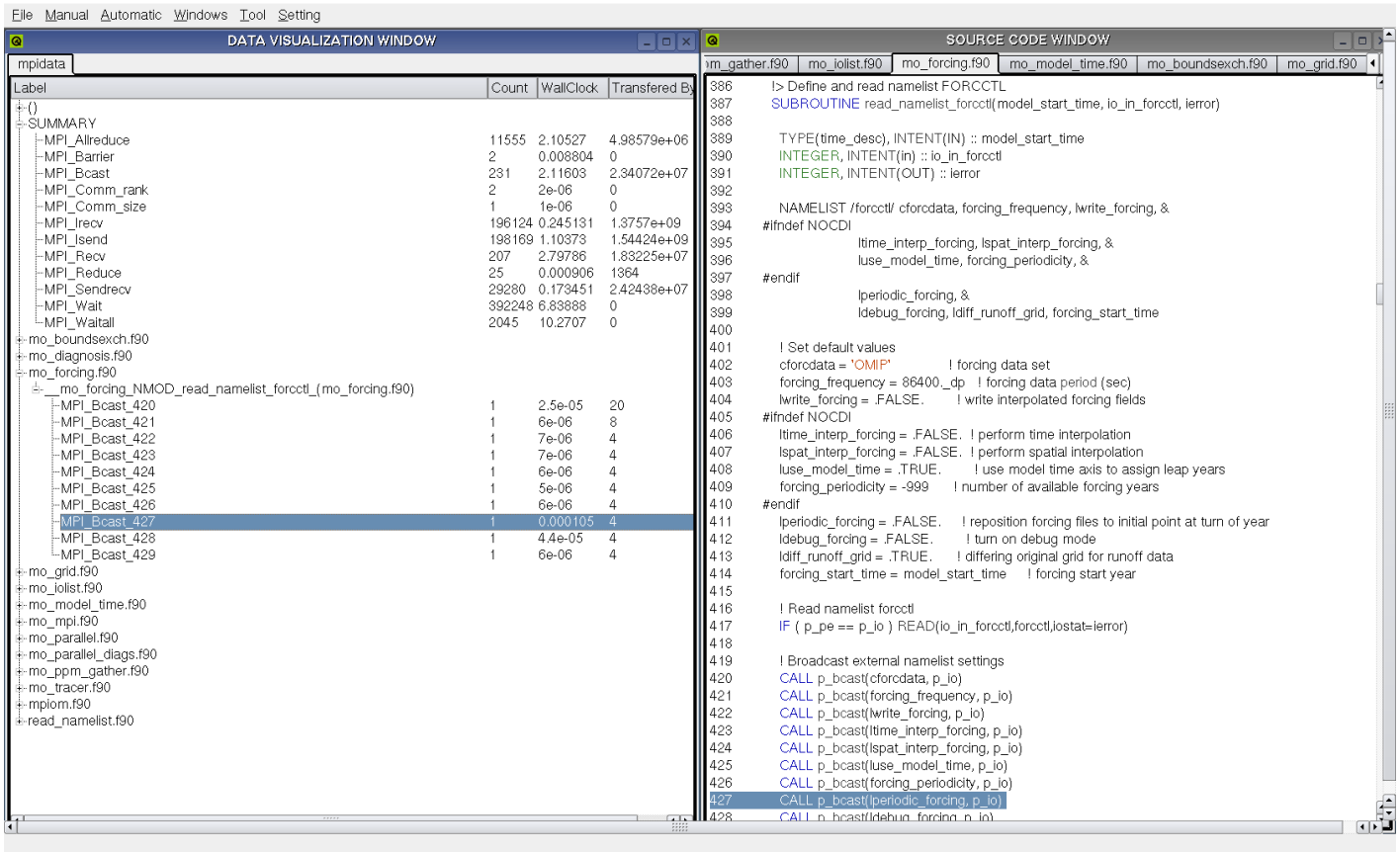
libmpitrace: profile

- default: data only for rank 0 and min, max, avg rank - set envVar `OUTPUT_ALL_RANKS=yes` for more
- `mpi_profile.<rank>` are human readable



libmpitrace: profile

view `mpi_profile_<rank>.viz` files with peekperf



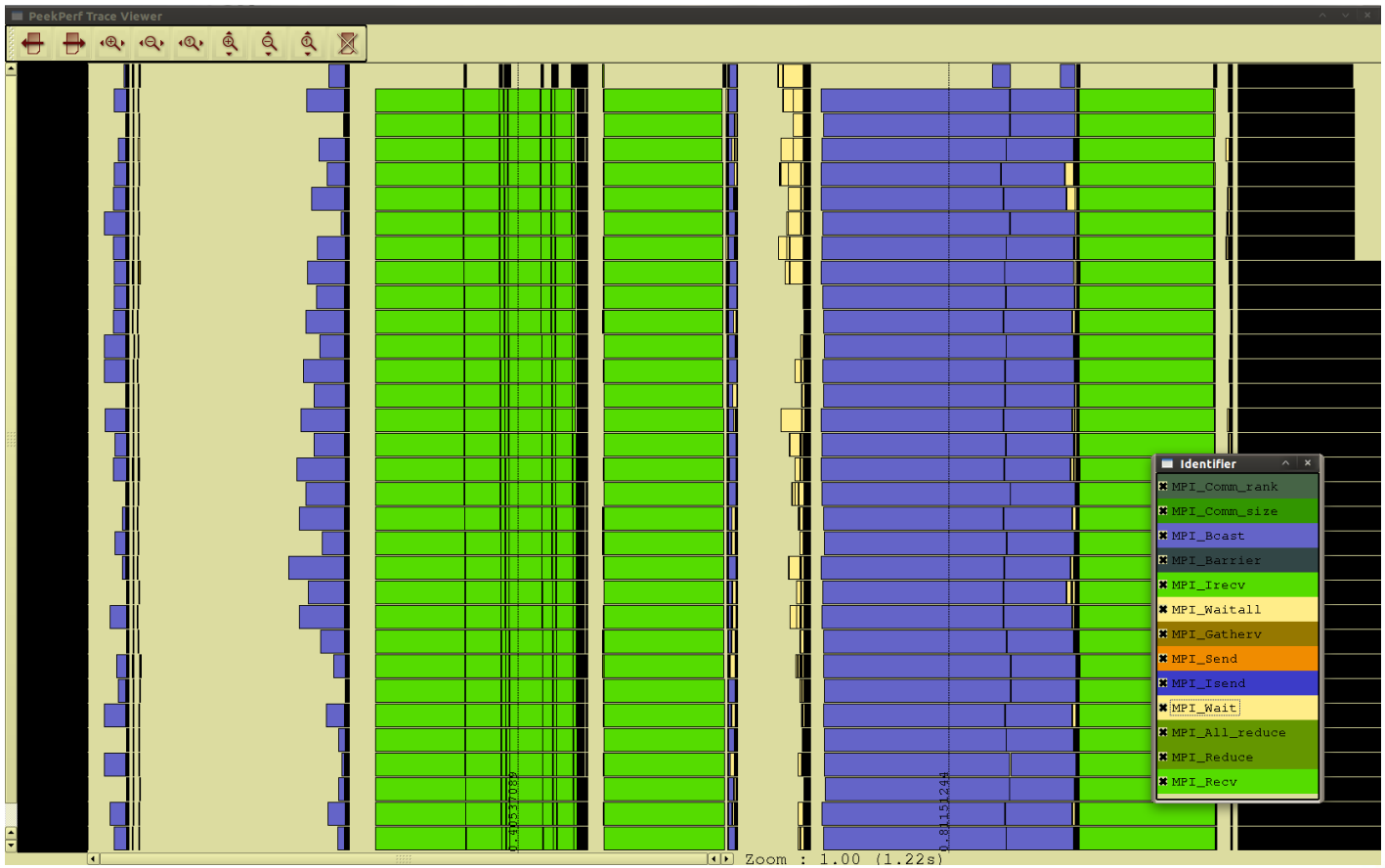
libmpitrace: customization

- `OUTPUT_ALL_RANKS=[yes|no]`: show all results
- `TRACE_ALL_EVENTS=[yes|no]`: do tracing or profiling
- `MAX_TRACE_RANK=#`: maximum of traced ranks
- `TRACEBACK_LEVEL=#`: useful for nested MPI-calls within other functions/libraries
- manual tracing for selected portion of the program through API
- profiling data cannot be controlled by the API - always collected throughout the entire execution of the program

→ `single_trace_0` outputfile contains tracing information from MPI ranks for which tracing was enabled

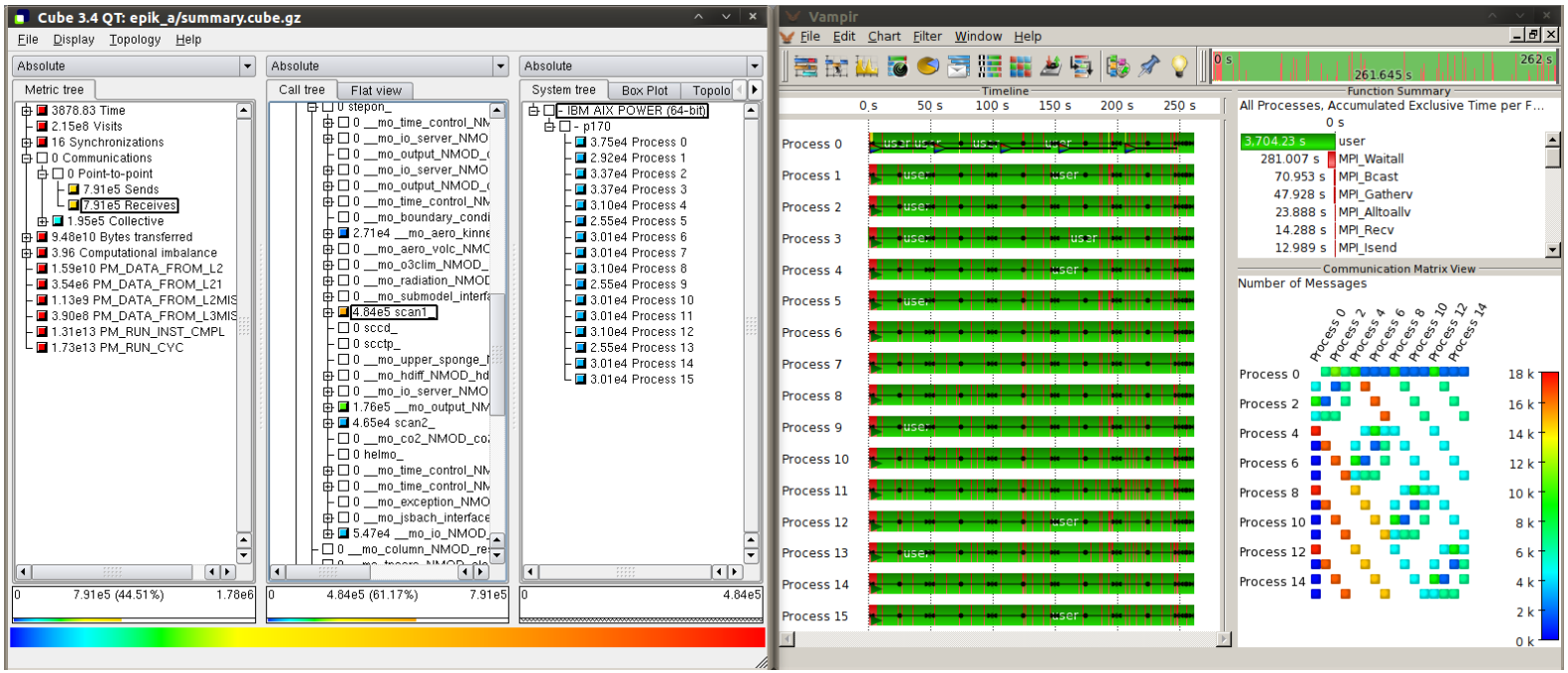
libmpitrace: trace

view `single_trace_0` file with peekview



Summary on basic tools

We have profiles of user subroutines and MPI calls by at least 2 different tools - one would be better
Additionally, the visualization could be better ...



If CPU time is not enough ...

Knowing hotspots of the application is only the beginning:

- why does some routine take so much time?
- why does MPI take so much time?

Answers: not trivial but often related to one of these issues

- parallelization aspects (imbalance, race-conditions, etc.)
  $\rightarrow$ covered by scalasca and vampir
- no suitable hardware usage (e.g. cache utilization)

If CPU time is not enough ...

... getting hardware counter data from IBM HPC Toolkit:
i/ `hpccount` command provides

- execution wall clock time
- resource utilization statistics
- hardware performance counters information
- derived hardware metrics

for the whole application run:

```
poe hpccount -u -n -o <name> <prog>
```

hpccount options

- -g specifies the hardware counter group
- -n suppresses output to stdout
- -o writes output to file <name>
- -u unique file names will be used

`HPM_ASC_OUTPUT, HPM_VIZ_OUTPUT` for ASCII or XML output
`HPM_AGGREGATE`:

- mirror.so: gets raw data from each MPI-task [default]
- average.so: counter groups distributed in round robin fashion! Aggregator takes avg over these subgroups

hpccount example

```
hpccount (IBM HPC Toolkit v5.2.2.4) summary

########  Resource Usage Statistics  ########

Total amount of time in user mode            : 119.653446 seconds
Total amount of time in system mode          : 0.053128 seconds
Maximum resident set size                    : 401968 Kbytes
Average shared memory use in text segment    : 7773 Kbytes*sec
Average unshared memory use in data segment  : 34396194 Kbytes*sec
Number of page faults without I/O activity   : 6925
Number of page faults with I/O activity      : 0
Number of times process was swapped out      : 0
Number of times file system performed INPUT  : 0
Number of times file system performed OUTPUT : 0
Number of IPC messages sent                  : 0
Number of IPC messages received              : 0
Number of signals delivered                  : 0
Number of voluntary context switches         : 493
Number of involuntary context switches       : 120

########  End of Resource Statistics  ########

Execution time (wall clock time)     : 121.634124139789 seconds

 PM_FPU_1FLOP (FPU executed one flop instruction)                 :        76000179
 PM_FPU_FMA (FPU executed multiply-add instruction)               :     16000000076
 PM_FPU_FSQRT_FDIV (FPU executed FSQRT or FDIV instruction)       :               8
 PM_FPU_FLOP (FPU executed 1FLOP, FMA, FSQRT or FDIV instruction) :     16076000263
 PM_RUN_INST_CMPL (Run instructions completed)                    :    151953175375
 PM_RUN_CYC (Run cycles)                                          :    564868441806

 Utilization rate                              :          98.724 %
 Instructions per run cycle                    :           0.269
 Total scalar floating point operations        :       32076.000 M
 Scalar flop rate (flops / WCT)                :         263.709 Mflop/s
 Scalar flops / user time                      :         267.116 Mflop/s
 Algebraic floating point operations           :       32076.000 M
 Algebraic flop rate (flops / WCT)             :         263.709 Mflop/s
 Algebraic flops / user time                   :         267.116 Mflop/s
 Scalar FMA percentage                         :          99.763 %
```

If CPU time is not enough ...

... getting hardware counter data from IBM HPC Toolkit:
ii/ instrumentation with libhpm

```
#include <libhpc.h>
int main(void) {
  ...
  MPI_Init(MPI_COMM_WORLD);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  hpmInit(myrank, "my_program");
  ... maybe some initialization code
  hpmStart(1, "first section");
  ... some code you want to analyze
  hpmStop(1);
  ... more boring code
  hpmTerminate(myrank);
  MPI_Finalize();
}
```

```
PROGRAM HELLO_WORLD
  IMPLICIT NONE
#include "f_hpc.h"
  ...
  CALL MPI_INIT(ierror)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierror)
  CALL f_hpminit(myrank, 'my_program')
  ... maybe some initialization code
  CALL f_hpmstart(1, 'first section')
  ... some code you want to analyze
  CALL f_hpmstop(1)
  ... more boring code
  CALL f_hpmterminate(myrank)
  CALL MPI_FINALIZE(ierror)
END
```

don't forget `-I/usr/lpp/ppe.hpct/include` and
`-L/usr/lpp/ppe.hpct/lib64 -lhpc -lpmapi`

Choosing counters

on POWER6

- you can measure 6 counters simultaneously
- not all combinations allowed
- 202 performance counter groups (`pmlist -g -1`)
- may need to sample multiple times for completeness
- or use multiplexing of counter groups

Use hpccount/libhpc to measure code efficiency by means of:

- Instructions per run cycle, Mflop/s (group 127)
- L1 cache usage (group 47)
- cache/memory access (group 7,11)

HPM counter analysis

counters are difficult to understand

- limited documentation
- experience needed to see counter value indicating a problem
- use IPC (instructions per run cycle) for first estimate

`IPC = PM_RUN_INST_CMPL / PM_RUN_CYC = 1/CPI`

category	IPC	description
1	< 0.4	Houston, we have a problem
2	$0.4 \ldots 0.7$	not tuned for POWER6
3	$0.7 \ldots 0.9$	acceptable
4	$0.9 \ldots 1.3$	very good (can be tough to get here)
5	> 1.3	wow ! (not always possible)
6	> 2.0	LINPACK, VMASS, ESSL, FFTW, ...

HPM counter analysis

group 127:

- `PM_FPU_1FLOP`: FPU executed single FLOP operation
- `PM_FPU_FMA`: FPU executed a multiply-add
- `PM_FPU_FSQRT_FDIV`
- `PM_FPU_FLOP`: 1flop, fma, sqrt, div operations for unit0 and 1

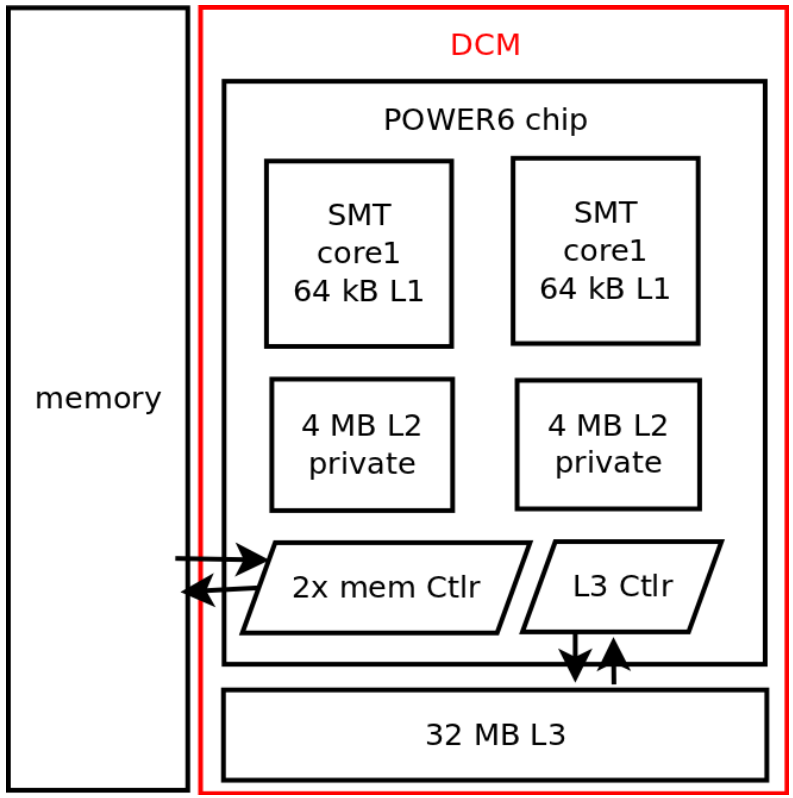
group 47:

- `PM_ST_REF_L1`: L1 D cache store references
- `PM_LD_REF_L1`: L1 D cache load references
- `PM_ST_MISS_L1`: L1 D cache store misses
- `PM_LD_MISS_L1`: L1 D cache load misses

HPM counter analysis

group 7:

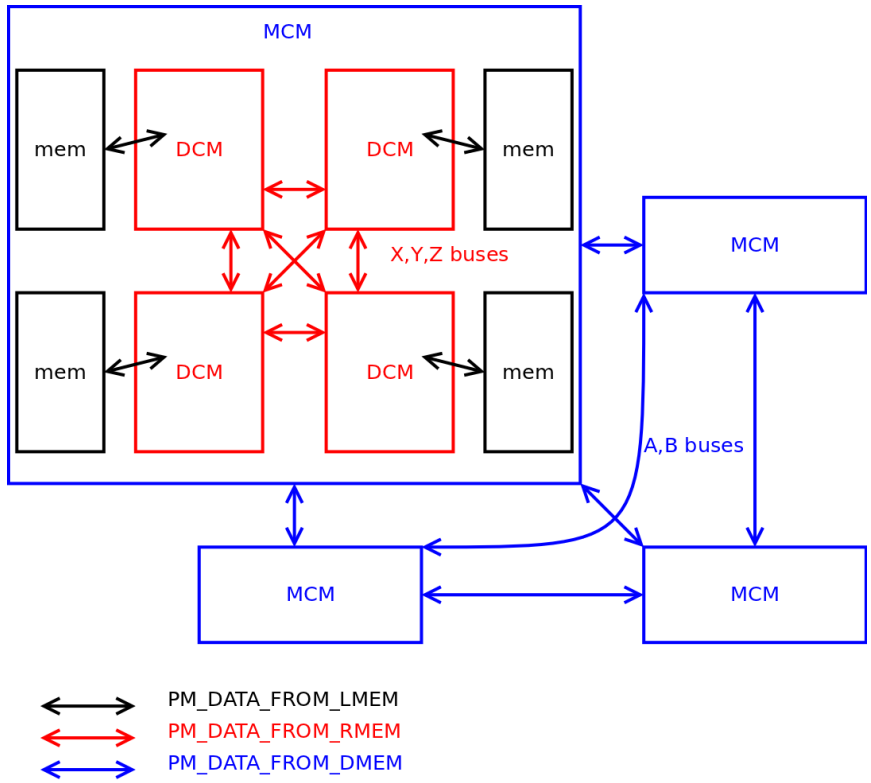
- `PM_DATA_FROM_L2`: D Cache reloaded from local L2
- `PM_DATA_FROM_L21`: D Cache reloaded from private L2 of other core on chip
- `PM_DATA_FROM_L2MISS`: D Cache reloaded but not from local L2
- `PM_DATA_FROM_L3MISS`: D Cache reloaded from beyond L3



HPM counter analysis

group 11:

- `PM_DATA_FROM_LMEM`: D Cache reloaded from local memory (attached to DCM)
- `PM_DATA_FROM_RMEM`: D Cache reloaded from remote memory (attached to other DCM on same MCM)
- `PM_DATA_FROM_DMEM`: D Cache reloaded from distant memory (attached to a different MCM)



HPCC results out of the box

i/ HPL: solves a dense linear system in double precision (linpack)
$N = 20000, NB = 20 \vee 120, P = 8, Q = 4$

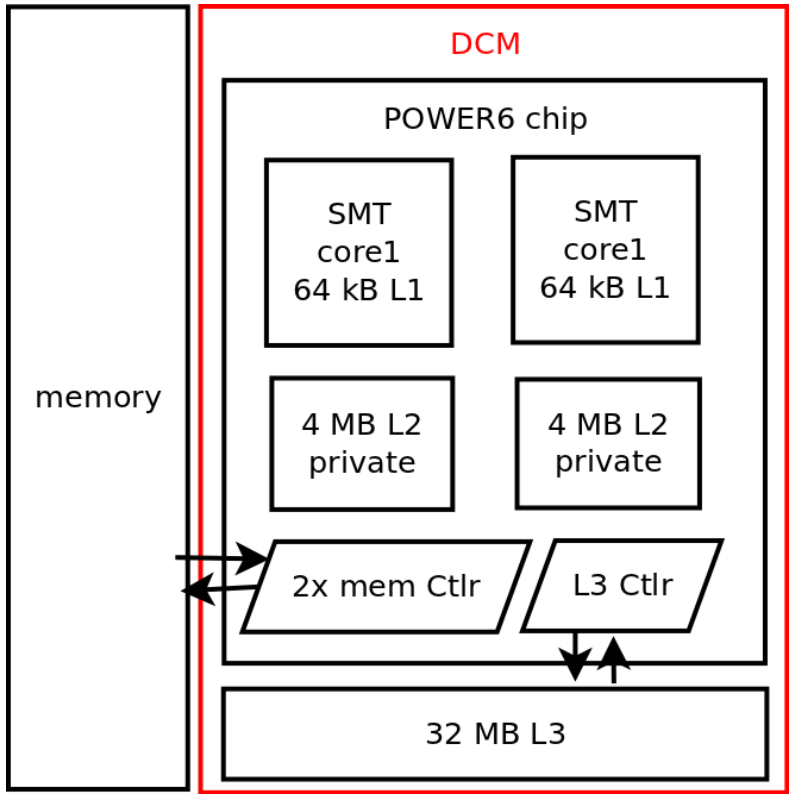
		libhpc: per MPI task			
NB	HPL Gflop/s	IPC	Gflop/s	% peak	L1 cache hit rate
20	2.39e+02	1.45 - 1.72	7.33 - 7.56	39.9 - 41.1 %	97.5 - 99.5 %
120	3.15e+02	1.82 - 2.09	9.75 - 9.8	52.3 - 56.0 %	98.5 - 99.4 %

	libhpc: per MPI task		
NB	PM_DATA_FROM_L2	PM_DATA_FROM_L2MISS	PM_DATA_FROM_LMEM
20	35131005 - 70195962	8913581 - 12180711	5921680 - 9110732
120	30105907 - 65539464	4992461 - 6045666	3332966 - 4227469

POWER6 background

POWER6 cache/memory: size and latency

- L1: 64 kB, 2 cycles
- L2: 4 MB core-private, 25 cycles
- L3: 32 MB chip-shared, 150 cycles
- mem: 50 (100) GB, 500 cycles



HPCC results out of the box

ii/ DGEMM: measures floating point rate of double precision real matrix-matrix multiplication
bench $N = 1500$ using libessl or not

essl	HPL Gflop/s (min/max)	libhpc: per MPI task IPC	libhpc: per MPI task Gflop/s
no	2.94 / 3.31	0.65 - 0.68	3.4 - 3.6
yes	10.84 / 15.36	2.49 - 2.59	14.2 - 14.8

essl	libhpc: per MPI task % peak	PM_ST_MISS_L1	PM_LD_MISS_L1
no	18.5 - 19.6 %	1.23e6 - 1.47e6	215.68e6 - 221.91e6
yes	77.3 - 80.6 %	2.71e6 - 3.32e6	0.93e6 - 1.99e6