

# Parallel Profiling using IPM on Rosa

Rosa Introduction Course

Scientific Computing Group

03 July 2009

# References

---

- IPM website (D. Skinner, K. Fuelberger, N. Wright)
  - <http://www.nersc.gov/nusers/resources/software/tools/ipm.php>
  - <http://ipm-hpc.sourceforge.net>
- CSCS website
  - **User Entry Point >**
    - **Software and Programming Environment >**
      - **Debugging and Performance Analysis > Performance > IPM**
- module help ipm

# IPM Features

---

- IPM is a profiling tool for MPI codes which :
  - Might work with your application when other tools don't
  - Is portable (CRAY, IBM, etc...), easy to use and Open Source
  - Provides a high-level performance summary of
    - MPI communication (time, # of calls, size, patterns)
    - Physical memory usage per compute node
    - Hardware performance counters (PAPI)
  - Is scalable, minimizes the impact on the running code, maintains a small memory footprint.
- IPM does not :
  - Support I/O (should be available soon),
  - Support function profiling,
  - Support OpenMP or PGAS languages.

# Using IPM regions

---

- Instrument your code

- `CALL MPI_Pcontrol (1, "Region_1\0")`

- ...

- `CALL MPI_Pcontrol (-1, "Region_1\0")`

- Separate IPM output generated for each defined region
- Non consecutive regions will be aggregated if same name used
- Non MPI code blocks can be defined

# Using IPM

- Profiling your code with IPM involves the following steps :
  - Load the IPM module
    - `module load ipm`
    - Using the CRAY wrappers (ftn, cc, CC) will automatically link your MPI code with the ipm library : no need to modify your compilation line, the wrapper will do it !
  - Relink your code ( No need to recompile ! )
    - `ftn -o test.exe test.f90`
    - Set PAPI counter group if needed : `export IPM_HPM=2`
  - Execute the resultant executable by submitting your PBS job as usual
    - `aprun -n 128 test.exe`
  - Visualize the results with the ipm\_parse perl script
    - `ipm_parse -help`
    - `ipm_parse -html username.12334535.324523.0`
      - can be done on your workstation !

# How to use IPM : basics

Upon completion you get

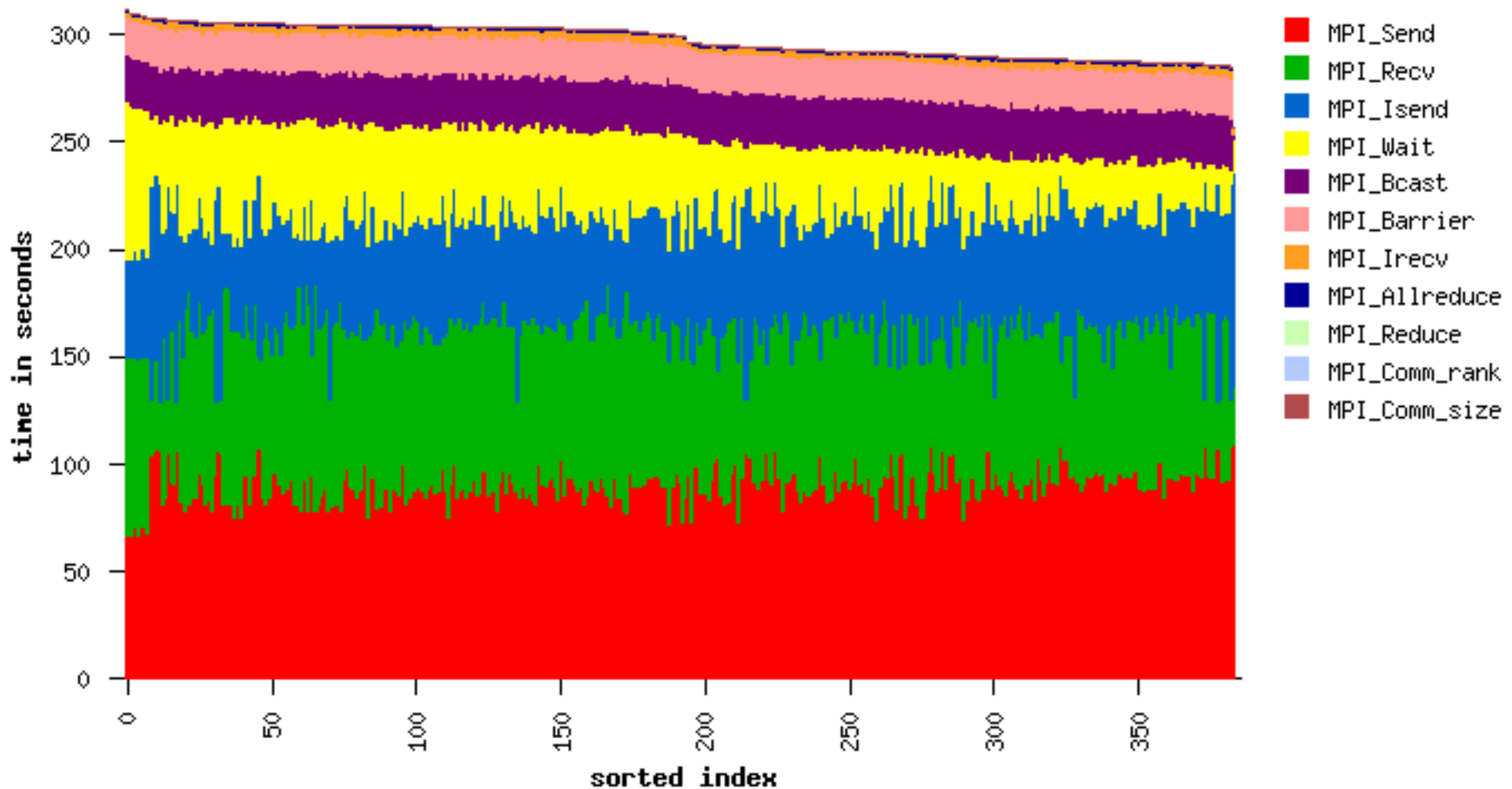
```
##IPMv0.85#####  
#  
# command : ../exe/pmemd -O -c inpcrd -o res (completed)  
# host      : s05405                mpi_tasks : 64 on 4 nodes  
# start     : 02/22/05/10:03:55     wallclock : 24.278400 sec  
# stop      : 02/22/05/10:04:17     %comm     : 32.43  
# gbytes    : 2.57604e+00 total      gflop/sec : 2.04615e+00 total  
#  
#####
```

Maybe that's enough.  
If so you're done.  
Have a nice day.

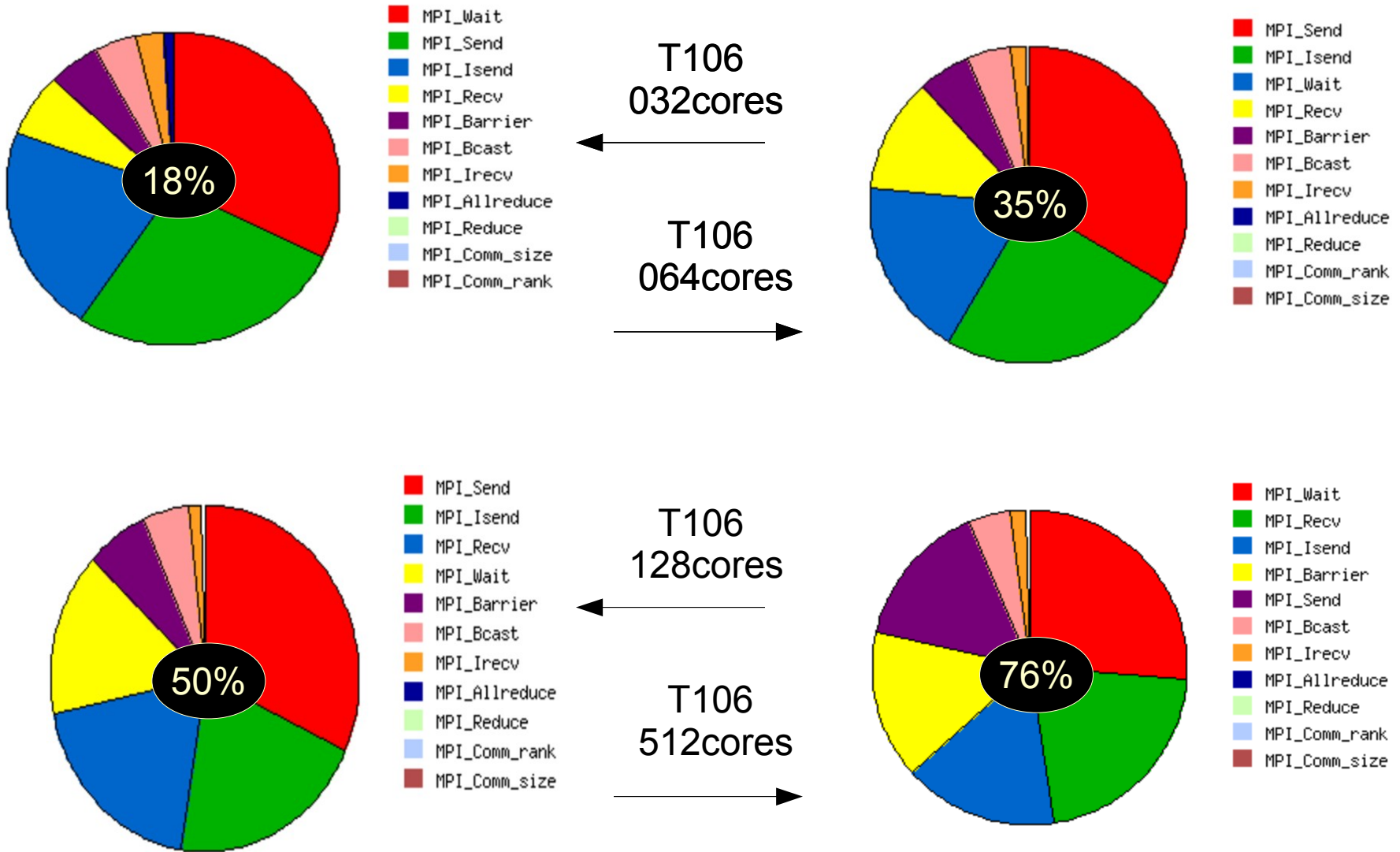
# Want more detail? IPM\_REPORT=full

# PM_CYC	3.00519e+11	4.69561e+09	4.50223e+09	5.83342e+09
# PM_FPU0_CMPL	2.45263e+10	3.83223e+08	3.3396e+08	5.12702e+08
# PM_FPU1_CMPL	1.48426e+10	2.31916e+08	1.90704e+08	2.8053e+08
# PM_FPU_FMA	1.03083e+10	1.61067e+08	1.36815e+08	1.96841e+08
# PM_INST_CMPL	3.33597e+11	5.21245e+09	4.33725e+09	6.44214e+09
# PM_LD_CMPL	1.03239e+11	1.61311e+09	1.29033e+09	1.84128e+09
# PM_ST_CMPL	7.19365e+10	1.12401e+09	8.77684e+08	1.29017e+09
# PM_TLB_MISS	1.67892e+08	2.62332e+06	1.16104e+06	2.36664e+07
#				
#	[time]	[calls]	<%mpi>	<%wall>
# MPI_Bcast	352.365	2816	69.93	22.68
# MPI_Waitany	81.0002	185729	16.08	5.21
# MPI_Allreduce	38.6718	5184	7.68	2.49
# MPI_Allgatherv	14.7468	448	2.93	0.95
# MPI_Isend	12.9071	185729	2.56	0.83
# MPI_Gatherv	2.06443	128	0.41	0.13
# MPI_Irecv	1.349	185729	0.27	0.09
# MPI_Waitall	0.606749	8064	0.12	0.04
# MPI_Gather	0.0942596	192	0.02	0.01
#####				

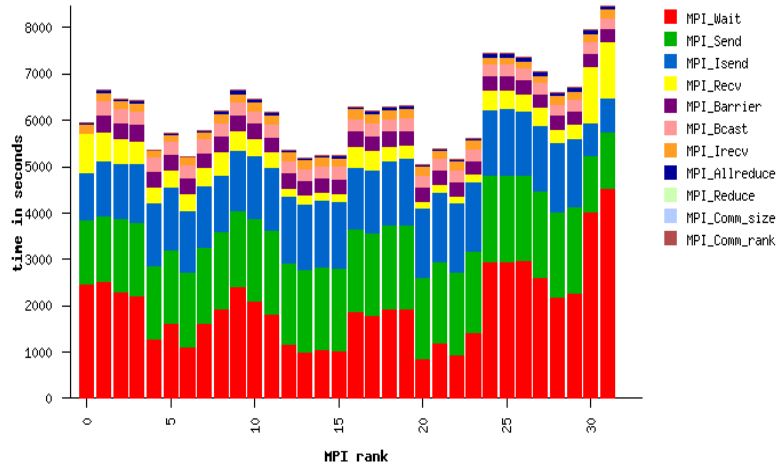
# Example IPM\_parse output (demo...)



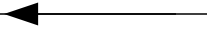
# ECHAM5 / XT5 : % of MPI Time



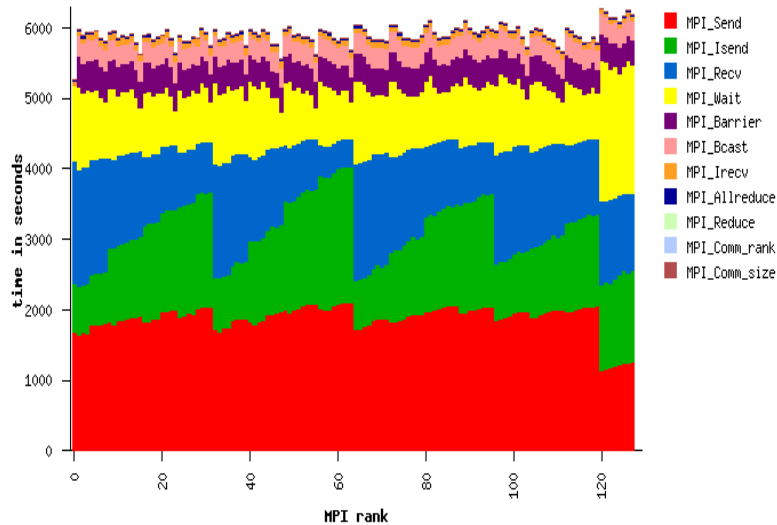
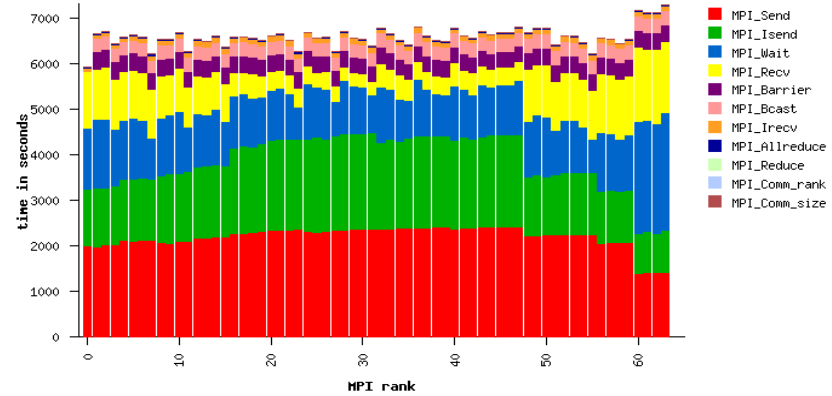
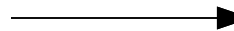
# ECHAM5 / XT5 : Communication balance by MPI rank



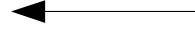
T106  
032cores



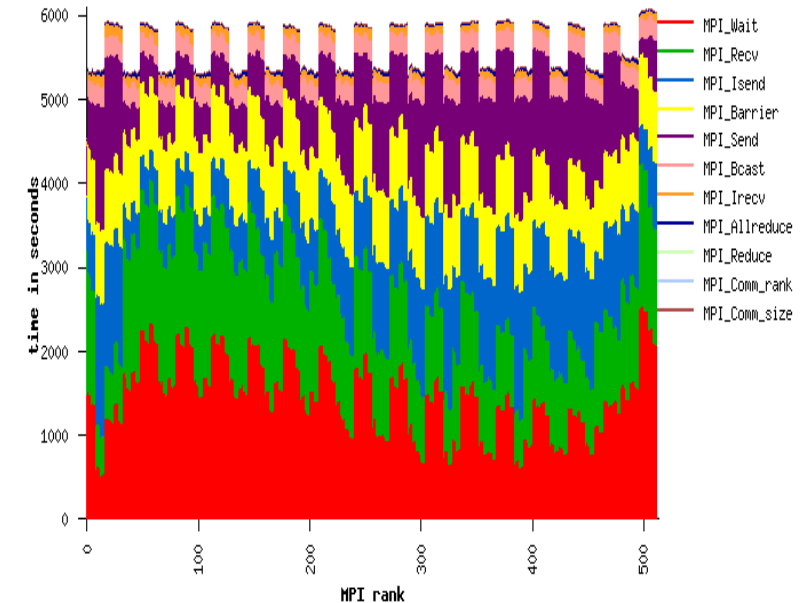
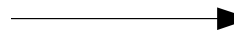
T106  
064cores



T106  
128cores



T106  
512cores



# CSCS CRAY XT5



- Rosa is a 20 cabinet Cray XT5 system :
  - 1844 compute nodes + 20 I/O nodes
  - 8cores per cnode => 14 752 compute cores @ 2.4 GHz Shanghai
  - 16 GBytes of memory per cnode => 29 TB
  - 9.6 GB/s interconnect bandwidth

\ PER	core	socket	node	blade	cabinet	rosa ( 19 SIO nodes Excluded )
core	-	4	8	32	768	14752
socket	-	-	2	8	192	3688
node	-	-	-	4	96	1844
blade	-	-	-	-	24	461
cabinet	-	-	-	-	-	19
<b>MEMORY (GB)</b>	2	8	16	64	1536	29504

# Want more detail? IPM\_REPORT=full

```
##IPMv0.85#####  
#  
# command : ../exe/pmemd -O -c inpcrd -o res (completed)  
# host      : s05405                mpi_tasks : 64 on 4 nodes  
# start     : 02/22/05/10:03:55     wallclock : 24.278400 sec  
# stop      : 02/22/05/10:04:17     %comm     : 32.43  
# gbytes    : 2.57604e+00 total      gflop/sec  : 2.04615e+00 total  
#  
#  
#           [total]           <avg>           min           max  
# wallclock      1373.67      21.4636      21.1087      24.2784  
# user           936.95       14.6398      12.68        20.3  
# system         227.7        3.55781     1.51         5  
# mpi            503.853       7.8727      4.2293      9.13725  
# %comm          32.4268      17.42        41.407  
# gflop/sec      2.04614      0.0319709   0.02724     0.04041  
# gbytes         2.57604      0.0402507   0.0399284   0.0408173  
# gbytes_tx      0.665125     0.0103926   1.09673e-05 0.0368981  
# gbyte_rx       0.659763     0.0103088   9.83477e-07 0.0417372  
#
```

## To extend IPM into a PAT, one needs...

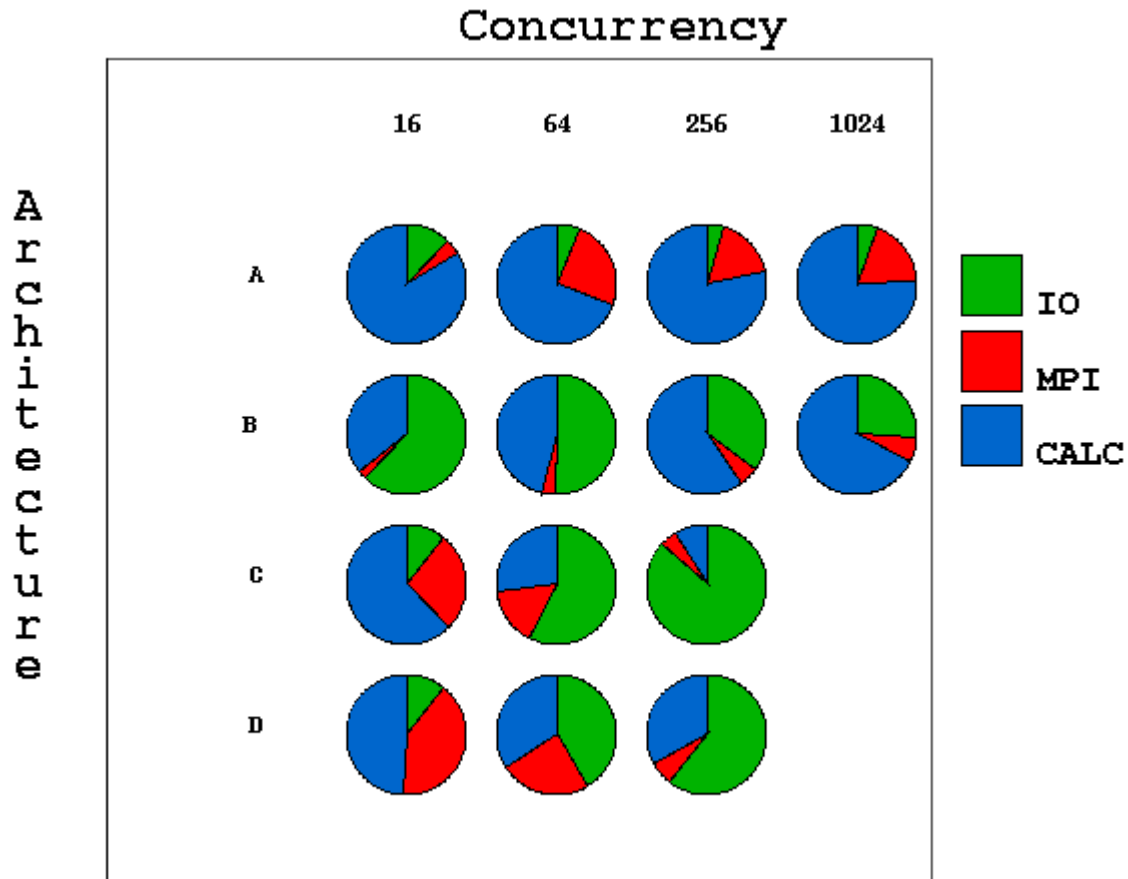
- ① More documentation
  - On its general usage/features
  - Available tools and analysis
- ② More post-processing tool development
  - Has not been a priority
- ③ IO and user-written code profiling
  - Some headway made on IO front
- ④ Additional visualization options
  - Developing new tools/software
  - Using existing software (CUBE, Walrus)
- ⑤ Additional parallelization paradigms
  - No OpenMP or PGAS support currently

# IPM: Methodology

- `MPI_Init()`
  - Initialize monitoring environment, allocate memory
- For each MPI call
  - Compute hash **key** from
    - Type of call (send/recv/bcast/...)
    - Buffer size (in bytes)
    - Communication partner rank
  - Store / update **value** in hash table with timing data
    - Number of calls,
    - minimum duration, maximum duration, summed time
- `MPI_Finalize()`
  - Aggregate, report to stdout, write XML log

# Portability: Profoundly Interesting

A high level description of the performance of a well known cosmology code on four well known architectures.



# IPM: XML log files

- There's a lot more information in the logfile than you get to stdout. A logfile is written that has the hash table, switch traffic, memory usage, executable information, ...
- Parallelism in writing of the log (when possible)
- The IPM logs are durable performance profiles serving
  - HPC center production needs: <https://www.nersc.gov/nusers/status/llsum/>  
[http://www.sdsc.edu/user\\_services/top/ipm/](http://www.sdsc.edu/user_services/top/ipm/)
  - HPC research: ipm\_parse renders txt and html  
<http://www.nersc.gov/projects/ipm/ex3/>
  - your own XML consuming entity, feed, or process