2023 Fall Parallel Programming
Get started with IPM & mpiP profiler

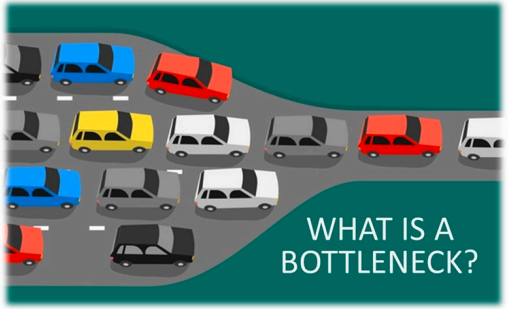
NTHU LSA-Lab

Profile your MPI program - IPM

IPM is a portable profiling tool for parallel codes, focusing on communication, computation, and IO. It offers low-overhead performance metrics for both production and development use in HPC centers. Runtime-adjustable detail levels are available through text and web reports.

Importance of Profiling

- Identify bottlenecks
- Understand data flow and communication patterns
- Optimize resource usage



Profile your MPI program - IPM

- `module load ipm/latest` : Load IPM module before you use
  - IPM Support collect MPI & MPIIO event and PAPI metrics
- IPM can be using in one of two modes either statically or dynamically:
  - Static way: In this case the users code needs to be relinked:
    - You could add this this after CFLAG or CXXFLAG in your Makefile

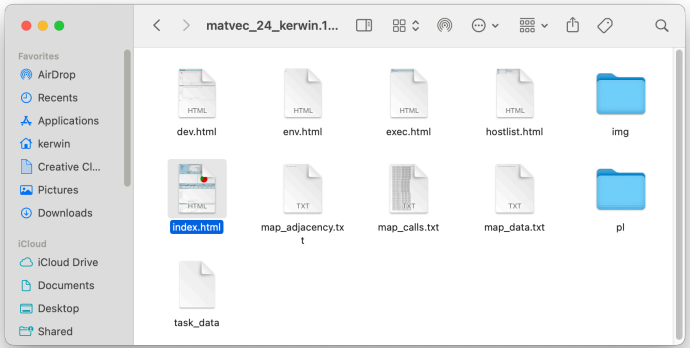
      ➡ `CFLAG = -O3 -lm –L/path/to/ipm/lib -lipm`

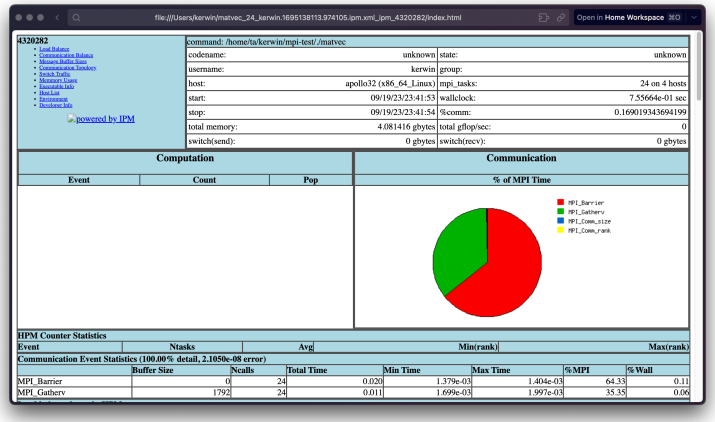
  - Dynamic way: `LD_PROLOAD` to attach profiler when you run your program

    ➡ `IPM_REPORT=full IPM_REPORT_MEM=yes IPM_LOG=full LD_PRELOAD=/opt/ipm/lib/libipm.so srun -n<process> ./<yout program>`

Profile your MPI program - IPM

- Output file will be <username>.xxxx.xml
- Using ipm_parse to convert the output file into html format
  - ➡ `ipm_parse -html <output_file>.ipm.xml`
- Your can download your output dir and open the html file on your computer.





IPM with Performance Application Programming Interface (PAPI)

The Performance Application Programming Interface (PAPI) is a library that provides a standard way to collect performance metrics from hardware components (CPU cycle, Instruction counts). It helps developers understand and optimize software performance in relation to the underlying hardware.

- Use `papi_avail` to dump all available metrics

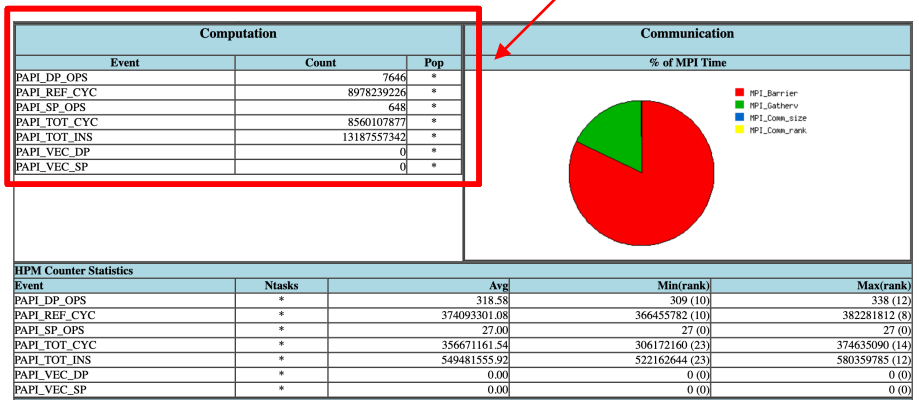
```
#!/bin/bash
#SBATCH -ptest
#SBATCH -N3
#SBATCH -n24

module load mpi

export IPM_REPORT=full
export IPM_REPORT_MEM=yes
export IPM_LOG=full
export LD_PRELOAD=/opt/ipm/lib/libipm.so
export IPM_HPM="PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_REF_CYC,\
PAPI_SP_OPS,PAPI_DP_OPS,PAPI_VEC_SP,PAPI_VEC_DP"

mpirun ./<your_program>
```

PAPI Event



Computation

Event	Count	Pop
PAPI_DP_OPS	7646	*
PAPI_REF_CYC	8978239226	*
PAPI_SP_OPS	648	*
PAPI_TOT_CYC	8560107877	*
PAPI_TOT_INS	13187557342	*
PAPI_VEC_DP	0	*
PAPI_VEC_SP	0	*

Communication

% of MPI Time

HPM Counter Statistics

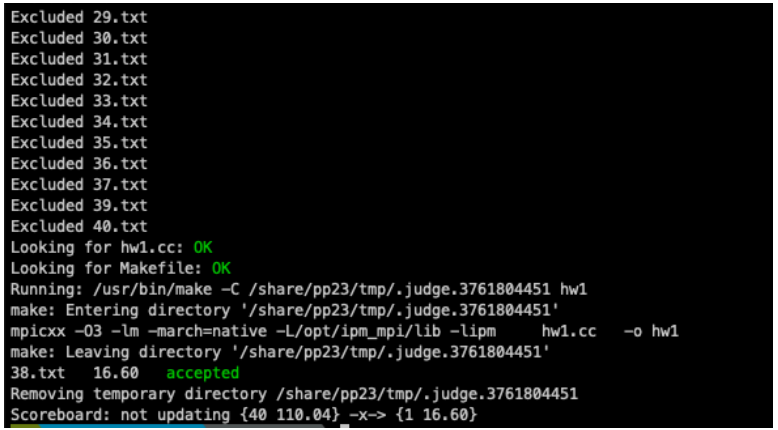
Event	Ntasks	Avg	Min(rank)	Max(rank)
PAPI_DP_OPS	*	318.58	309 (10)	338 (12)
PAPI_REF_CYC	*	374093301.08	366455782 (10)	382281812 (8)
PAPI_SP_OPS	*	27.00	27 (0)	27 (0)
PAPI_TOT_CYC	*	356671161.54	306172160 (23)	374635090 (14)
PAPI_TOT_INS	*	549481555.92	522162644 (23)	580359785 (12)
PAPI_VEC_DP	*	0.00	0 (0)	0 (0)
PAPI_VEC_SP	*	0.00	0 (0)	0 (0)

In this case, we highly recommend using sbatch to submit jobs.
Here is the sbatch script example !!

LSA-Lab

Using judge unit test

- Using hw1-judge -i <testcase e.g. 01.txt> to run single case judge.
  - It only works with IPM in static mode.
  - Use 'export' or 'setenv' for IPM_HPM (a shell variable).
  - Don’t forget to delete IPM linking argument when your submit your code. Profiling must cause overhead in your program.



Profile your MPI program - mpiP

mpiP is a light-weight profiling library for MPI applications. Because it only collects statistical information about MPI functions, mpiP generates considerably less overhead and much less data than tracing tools. All the information captured by mpiP is task-local. It only uses communication during report generation, typically at the end of the experiment, to merge results from all of the tasks into one output file.

```
#!/bin/bash
#SBATCH -ptest
#SBATCH -N3
#SBATCH -n24

module load mpi

export MPIP="-y -l"
export LD_PRELOAD=/opt/mpiP/lib/libmpiP.so

mpirun ./<your_program>
```

```
[kerwin@apollo31 hw1]$ ls -al
total 2097356
drwxr-xr-x  2 kerwin kerwin        290 Sep 20 14:48 .
drwxr-xr-x 12 kerwin kerwin        213 Sep 15 04:17 ..
-rw-r--r--  1 kerwin kerwin          0 Sep 12 15:44 4318744.err
-rw-r--r--  1 kerwin kerwin          0 Sep 12 15:44 4318744.log
-rwxr-xr-x  1 kerwin kerwin      31464 Sep 18 03:31 hw1
-rw-r--r--  1 kerwin kerwin      40094 Sep 19 13:18 hw1.12.5419.1.mpiP
-rw-r--r--  1 kerwin kerwin      47420 Sep 18 22:30 hw1.15.4007.1.mpiP
-rw-r--r--  1 kerwin kerwin      13216 Nov 21  2021 hw1.cc
-rw-r--r--  1 kerwin kerwin      46449 Sep 19 23:31 kerwin.1695137444.992549.ipm.xml
-rw-r--r--  1 kerwin kerwin        160 Sep 15 17:45 Makefile
-rwxr-xr-x  1 kerwin kerwin        184 Sep 18 02:59 multi.sh
-rw-r--r--  1 kerwin kerwin 2147479552 Sep 19 23:31 out
-rwxr-xr-x  1 kerwin kerwin       7714 Sep  6 16:30 runner.py
-rw-r--r--  1 kerwin kerwin        712 Sep 12 15:44 sbatch.sh
```

Output file will be *.mpiP

LSA-Lab

Profile your MPI program - Understanding mpiP output report

- **Header information**
  - provides basic information about your performance experiment.
  - MPI_Time (by each processes) and Callsites.
- **Agregate information**
  - Including Time, Message Size, Collective time, P2P sent size and some statistics.
- **Callsites information**
  - I/O statistics and more....
- Revise the online document to understand the meaning of each section.

About homework report

- Homework 中我們希望同學透過 Profile 的方法來確定自己實作的平行程式運行效能，以及用來佐證在優化方法是如何改進性能。
- 我們有兩個 Profiler ，IPM 會幫你繪製部分圖表，而 mpiP 則是將資料變成類似表格的方式儲存，同學可以利用任何資料視覺化的方法將這些結果呈現在 report 內。
- 另外關於 load balance 的部分下面提供一個範例，左圖為不 balance 的 MPI program 而右圖則比較 balance。
- 相比 CPU File IO 的誤差會比較明顯，同學可以活用 debug print 和觀察 profiler output 中的 MPI IO API 被 call 的次數來理解 IO 的表現。
- Home 目錄和 judge 時所用的 disk 不同因此速度有很大的差異，如果想要觀察 MPI IO 的 performance 可以使用前面提到的 judge unit test + IPM static mode 來達成效果。
- Profiler 有 overhead ，Scoreboard 上的數字也不是絕對。 本次作業 report 分數佔 30% ，希望同學用心寫 report。