Using PAPI to Monitor Communication Between Processes Running in Parallel

Daniel Topa

April 25, 2025

Abstract

This document demonstrates how to use the **Performance Application Programming Interface (PAPI)** to profile inter-process communication in parallel applications. We cover integration with **MPI**, **Coarray Fortran**, and **Legion**, compare PAPI with alternative tools, and provide ready-to-use code examples. Includes installation via Spack and advanced profiling toolchains.

1 Installation

1.1 Installing PAPI via Spack

```
spack install papi
```

Listing 1: Installing PAPI with Spack

2 Monitoring MPI Communication

```c
#include <mpi.h>
#include <papi.h>

int main(int argc, char **argv) {
    MPI_Init(&argc, &argv);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Initialize PAPI
    int events[2] = {PAPI_TOT_CYC, PAPI_L2_TCM};
    long long values[2];
    PAPI_start_counters(events, 2);

    // Communication pattern
    double data[1000];
    if (rank == 0) {
        MPI_Send(data, 1000, MPI_DOUBLE, 1, 0, MPI_COMM_WORLD);
    } else {
        MPI_Recv(data, 1000, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
    }

    // Read counters
    PAPI_stop_counters(values, 2);
    printf("Rank %d: Cycles=%lld, L2 Misses=%lld\n",
           rank, values[0], values[1]);

```

```
    MPI_Finalize();
    return 0;
}
```

Listing 2: MPI+PAPI Example

2.1 Compilation and Execution

```
mpicc -o mpi_papi mpi_papi.c -lpapi
mpiexec -n 2 ./mpi_papi
```

2.2 Expected Output

```
Rank 0: Cycles=1200000, L2 Misses=850
Rank 1: Cycles=1800000, L2 Misses=920
```

3 Monitoring Coarray Fortran

3.1 Example Code

```
program caf_papi
  use iso_c_binding
  implicit none
  integer(c_int) :: me, np
  real :: array[*]

  interface
    subroutine papi_start() bind(C)
    end subroutine
    function papi_read_cycles() bind(C) result(cycles)
      import :: c_long_long
      integer(c_long_long) :: cycles
    end function
  end interface

  me = this_image()
  np = num_images()

  call papi_start()
  array = me * 2.0
  sync all  ! Measure synchronization cost
  print *, "Image", me, "Cycles:", papi_read_cycles()
end program
```

Listing 3: Coarray+PAPI Example

3.2 Compilation and Execution

```
caf -lpapi caf_papi.f90 -o caf_papi
cafrun ./caf_papi  # assumes chmod +x caf_papi
```

4 Complementary Profiling Tools

4.1 Tracing Tools

- **VampirTrace/Score-P**: Combines MPI, OpenMP, and Coarray Fortran tracing

```
spack install scorep
export SCOREP_ENABLE_TRACING=1
scorep --caf mpif90 -o caf_prog caf_prog.f90
```

- **TAU (Tuning and Analysis Utilities)**:

```
spack install tau
tau_f90.sh -o caf_tau caf_prog.f90
```

4.2 Call-Path Profiling

- **HPCToolkit**:

```
spack install hpctoolkit
hpcrun -e PAPI_TOT_CYC ./caf_prog
hpcprof -S caf_prog.hpcstruct hpctoolkit-caf_prog-*
```

- **Legion Prof**:
  - Built into Legion runtime (enable with `-lg:prof`)
  - Visualizes task graphs and dependencies
  - Compatible with PAPI counters via `-lg:papi`

5 Tool Comparison

Tool	Strengths	Integration
PAPI	Hardware counters	Manual instrumentation
Score-P	Full MPI/CAF tracing	Automated via compiler
TAU	Portable profiling	Runtime configuration
HPCToolkit	Call-path analysis	Post-processing
Legion Prof	Task visualization	Native to Legion

Table 1: Advanced Profiling Tools Comparison

```
# Run Legion application with PAPI
LEGION_PAPI_COUNTERS=PAPI_TOT_CYC,PAPI_FP_OPS ./legion_prog -lg:papi 1
```

Listing 4: Legion with PAPI Profiling

Appendix A: Spack Package Information

PAPI Package Details

The following shows the output of `spack info papi`, which provides comprehensive information about available versions, variants, and dependencies:

```
==> Warning: The packages:all:compiler preference has been deprecated...
AutotoolsPackage:   papi

Description:
    PAPI provides the tool designer and application engineer with a
    consistent interface and methodology for use of the performance counter
    hardware found in most major microprocessors...
```

```
Homepage: https://icl.utk.edu/papi/

Preferred version:
    7.1.0      https://icl.utk.edu/projects/papi/downloads/papi-7.1.0.tar.gz

Safe versions:
    master     [git] https://github.com/icl-utk-edu/papi on branch master
    7.1.0      https://icl.utk.edu/projects/papi/downloads/papi-7.1.0.tar.gz
    7.0.1      https://icl.utk.edu/projects/papi/downloads/papi-7.0.1.tar.gz
    6.0.0.1    https://icl.utk.edu/projects/papi/downloads/papi-6.0.0.1.tar.gz
    [additional versions...]

Variants:
    build_system [autotools]        autotools
    cuda [false]                    false, true
    debug [false]                   false, true
    example [true]                  false, true
    infiniband [false]              false, true
    lmsensors [false]               false, true
    nvml [false]                    false, true
    powercap [false]                false, true
    rapl [false]                    false, true
    rocm [false]                    false, true
    rocm_smi [false]                false, true
    sde [false]                     false, true
    shared [true]                   false, true
    static_tools [false]            false, true

    when +rocm
      amdgpu_target [none]          none, gfx1010, gfx1011, [additional targets...]

    when @6.0.0:
      rdpmc [true]                  false, true

Build Dependencies:
    bc  c  cuda  cxx  fortran  gmake  gnuconfig  hip  hsa-rocr-dev  llvm-amdgpu
    lm-sensors  rocm-openmp-extras  rocm-smi-lib  rocprofiler-dev

Link Dependencies:
    cuda  hip  hsa-rocr-dev  llvm-amdgpu  lm-sensors  rocm-openmp-extras
    rocm-smi-lib  rocprofiler-dev

Run Dependencies:
    None

Licenses:
    BSD-3-Clause
```

Key observations:

- Latest stable version: 7.1.0
- Important variants: `debug`, `cuda`, `rocm` for GPU support
- ROCm requires specific AMD GPU architecture specification
- BSD-3-Clause license allows flexible use