

# MPIP

## A Simple Profiling Tool for MPI

---

Martin Schulz<sup>1</sup>

(originally tool by LLNL and ORNL, maintainer Chris Chambreau, LLNL)

---

<sup>1</sup> TU Munich


## So, You Need to Look at a New Application ...

---

Scenarios:

- New application development
- Analyze/Optimize external application
- Suspected bottlenecks

First goal: overview of ...

- Communication frequency and intensity
  - Types and complexity of communication
  - Source code locations of expensive MPI calls
  - Differences between processes
- 

## Basic Principle of Profiling MPI

---

Intercept all MPI API calls

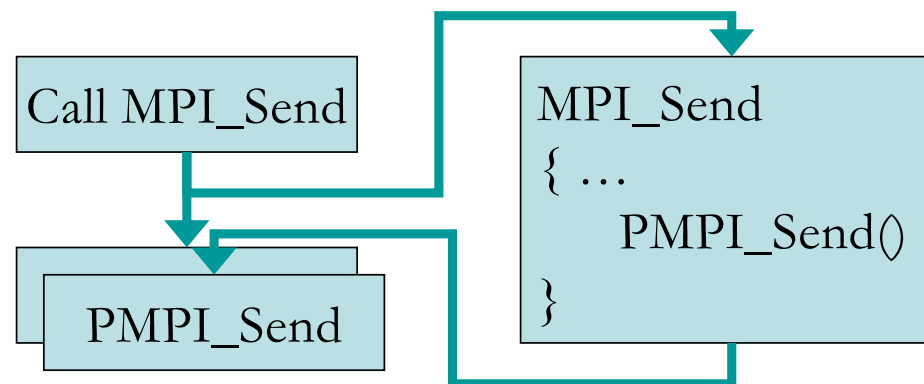
- Using wrappers for all MPI calls

Aggregate statistics over time

- Number of invocations
- Data volume
- Time spent during function execution

Multiple aggregations options/granularity

- By function name or type
- By source code location (call stack)
- By process rank



## mpiP: Efficient MPI Profiling

---

Open source MPI profiling library

- Developed at LLNL, maintained by LLNL & ORNL
- Available from sourceforge
- Works with any MPI library

Easy-to-use and portable design

- Relies on PMPI instrumentation
- No additional tool daemons or support infrastructure
- Single text file as output
- Optional: GUI viewer

## Running with mpiP 101 / Experimental Setup

---

mpiP works on binary files

- Uses standard development chain
- Use of “-g” recommended

Run option 1: Relink

- Specify libmpi.a/.so on the link line
- Portable solution, but requires object files

Run option 2: library preload

- Set preload variable (e.g., LD\_PRELOAD) to mpiP
- Transparent, but only on supported systems

## Running with mpiP 101 / Running

```
bash-3.2$ srun -n4 smg2000
mpiP:
mpiP:
mpiP: mpiP V3.1.2 (Build Dec 16 2008/17:31:26)
mpiP: Direct questions and errors to mpiP-
mpiP: help@lists.sourceforge.net
mpiP:
Running with these driver parameters:
(nx, ny, nz) = (60, 60, 60)
(Px, Py, Pz) = (4, 1, 1)
(bx, by, bz) = (1, 1, 1)
(cx, cy, cz) = (1.000000, 1.000000, 1.000000)
(n_pre, n_post) = (1, 1)
dim = 3
solver ID = 0
```

```
=====
Struct Interface:
=====
```

```
Struct Interface:
```

```
wall clock time = 0.075800 seconds
cpu clock time = 0.080000 seconds
```

Header

```
=====
Setup phase times:
=====
```

```
SMG Setup:
```

```
wall clock time = 1.473074 seconds
cpu clock time = 1.470000 seconds
=====
```

```
Solve phase times:
=====
```

```
SMG Solve:
```

```
wall clock time = 8.176930 seconds
cpu clock time = 8.180000 seconds
```

```
Iterations = 7
```

```
Final Relative Residual Norm = 1.459319e-07
```

Output File

```
mpiP:
mpiP: Storing mpiP output in [./smg2000-p.4.11612.1.mpiP].
mpiP:
bash-3.2$
```

## mpiP 101 / Output – Metadata

```
@ mpiP
@ Command : ./smg2000-p -n 60 60 60
@ Version      : 3.1.2
@ MPIP Build date : Dec 16 2008, 17:31:26
@ Start time   : 2009 09 19 20:38:50
@ Stop time    : 2009 09 19 20:39:00
@ Timer Used   : gettimeofday
@ MPIP env var  : [null]
@ Collector Rank : 0
@ Collector PID : 11612
@ Final Output Dir : .
@ Report generation : Collective
@ MPI Task Assignment : 0 hera27
@ MPI Task Assignment : 1 hera27
@ MPI Task Assignment : 2 hera31
@ MPI Task Assignment : 3 hera31
```

## mpiP 101 / Output – Overview

@--- MPI Time (seconds) ---

Task	AppTime	MPITime	MPI%
0	9.78	1.97	20.12
1	9.8	1.95	19.93
2	9.8	1.87	19.12
3	9.77	2.15	21.99
*	39.1	7.94	20.29



## mpiP 101 / Output – Callsites

@--- Callsites: 23 ---

ID	Lev	File/Address	Line	Parent_Funct	MPI_Call
1	0	communication.c	1405	hypr_CommPkgUnCommit	Type_free
2	0	timing.c	419	hypr_PrintTiming	Allreduce
3	0	communication.c	492	hypr_InitializeCommunication	Isend
4	0	struct_innerprod.c	107	hypr_StructInnerProd	Allreduce
5	0	timing.c	421	hypr_PrintTiming	Allreduce
6	0	coarsen.c	542	hypr_StructCoarsen	Waitall
7	0	coarsen.c	534	hypr_StructCoarsen	Isend
8	0	communication.c	1552	hypr_CommTypeEntryBuildMPI	Type_free
9	0	communication.c	1491	hypr_CommTypeBuildMPI	Type_free
10	0	communication.c	667	hypr_FinalizeCommunication	Waitall
11	0	smg2000.c	231	main	Barrier
12	0	coarsen.c	491	hypr_StructCoarsen	Waitall
13	0	coarsen.c	551	hypr_StructCoarsen	Waitall
14	0	coarsen.c	509	hypr_StructCoarsen	Irecv
15	0	communication.c	1561	hypr_CommTypeEntryBuildMPI	Type_free
16	0	struct_grid.c	366	hypr_GatherAllBoxes	Allgather
17	0	communication.c	1487	hypr_CommTypeBuildMPI	Type_commit
18	0	coarsen.c	497	hypr_StructCoarsen	Waitall
19	0	coarsen.c	469	hypr_StructCoarsen	Irecv
20	0	communication.c	1413	hypr_CommPkgUnCommit	Type_free
21	0	coarsen.c	483	hypr_StructCoarsen	Isend
22	0	struct_grid.c	395	hypr_GatherAllBoxes	Allgatherv
23	0	communication.c	485	hypr_InitializeCommunication	Irecv

## mpiP 101 / Output – per Function Timing

@--- Aggregate Time (top twenty, descending, milliseconds) ---

Call	Site	Time	App%	MPI%	COV
Waitall	10	4.4e+03	11.24	55.40	0.32
Isend	3	1.69e+03	4.31	21.24	0.34
Irecv	23	980	2.50	12.34	0.36
Waitall	12	137	0.35	1.72	0.71
Type_commit	17	103	0.26	1.29	0.36
Type_free	9	99.4	0.25	1.25	0.36
Waitall	6	81.7	0.21	1.03	0.70
Type_free	15	79.3	0.20	1.00	0.36
Type_free	1	67.9	0.17	0.85	0.35
Type_free	20	63.8	0.16	0.80	0.35
Isend	21	57	0.15	0.72	0.20
Isend	7	48.6	0.12	0.61	0.37
Type_free	8	29.3	0.07	0.37	0.37
Irecv	19	27.8	0.07	0.35	0.32
Irecv	14	25.8	0.07	0.32	0.34
...					

## mpiP 101 / Output – per Function Message Size

@--- Aggregate Sent Message Size (top twenty, descending, bytes) ---@

Call	Site	Count	Total	Avrg	Sent%
Isend	3	260044	2.3e+08	885	99.63
Isend	7	9120	8.22e+05	90.1	0.36
Isend	21	9120	3.65e+04	4	0.02
Allreduce	4	36	288	8	0.00
Allgather	22	4	112	28	0.00
Allreduce	2	12	96	8	0.00
Allreduce	5	12	96	8	0.00
Allgather	16	4	16	4	0.00

## mpiP 101 / Output – per Callsite Timing

```
-----  
@--- Callsite Time statistics (all, milliseconds): 92 -----  
-----
```

Name	Site	Rank	Count	Max	Mean	Min	App%	MPI%
Allgather	16	0	1	0.034	0.034	0.034	0.00	0.00
Allgather	16	1	1	0.049	0.049	0.049	0.00	0.00
Allgather	16	2	1	2.92	2.92	2.92	0.03	0.16
Allgather	16	3	1	3	3	3	0.03	0.14
Allgather	16	*	4	3	1.5	0.034	0.02	0.08
Allgatherv	22	0	1	0.03	0.03	0.03	0.00	0.00
Allgatherv	22	1	1	0.036	0.036	0.036	0.00	0.00
Allgatherv	22	2	1	0.022	0.022	0.022	0.00	0.00
Allgatherv	22	3	1	0.022	0.022	0.022	0.00	0.00
Allgatherv	22	*	4	0.036	0.0275	0.022	0.00	0.00
Allreduce	2	0	3	0.382	0.239	0.011	0.01	0.04
Allreduce	2	1	3	0.31	0.148	0.046	0.00	0.02
Allreduce	2	2	3	0.411	0.178	0.062	0.01	0.03
Allreduce	2	3	3	1.33	0.622	0.062	0.02	0.09
Allreduce	2	*	12	1.33	0.297	0.011	0.01	0.04

```
...
```

## mpiP 101 / Output – per Callsite Message Size

```
-----  
@--- Callsite Message Sent statistics (all, sent bytes) -----  
-----
```

Name	Site	Rank	Count	Max	Mean	Min	Sum
Allgather	16	0	1	4	4	4	4
Allgather	16	1	1	4	4	4	4
Allgather	16	2	1	4	4	4	4
Allgather	16	3	1	4	4	4	4
Allgather	16	*	4	4	4	4	16
Allgatherv	22	0	1	28	28	28	28
Allgatherv	22	1	1	28	28	28	28
Allgatherv	22	2	1	28	28	28	28
Allgatherv	22	3	1	28	28	28	28
Allgatherv	22	*	4	28	28	28	112
Allreduce	2	0	3	8	8	8	24
Allreduce	2	1	3	8	8	8	24
Allreduce	2	2	3	8	8	8	24
Allreduce	2	3	3	8	8	8	24
Allreduce	2	*	12	8	8	8	96

```
...
```

## Fine Tuning the Profile Run

---

### mpiP Advanced Features

- User controlled stack trace depth
- Reduced output for large scale experiments
- Application control to limit scope
- Measurements for MPI I/O routines

### Controlled by MPIP environment variable

- Set by user before profile run
- Command line style argument list
- Example: MPIP = "-c -o -k 4"

## mpiP Parameters

Param.	Description	Default
-c	Concise Output / No callsite data	
-f dir	Set output directory	
-k n	Set callsite stack traceback size to n	1
-l	Use less memory for data collection	
-n	Do not truncate pathnames	
-o	Disable profiling at startup	
-s n	Set hash table size	256
-t x	Print threshold	0.0
-v	Print concise & verbose output	

## Controlling the Stack Trace

---

Callsites are determined using stack traces

- Starting from current call stack going backwards
- Useful to avoid MPI wrappers
- Helps to distinguish library invocations

Tradeoff: stack trace depth

- Too short: can't distinguish invocations
- Too long: extra overhead / too many call sites

User can set stack trace depth

- -k <n> parameter



## Concise Output

---

Output file contains many details

- Users often only interested in summary
- Per callsite/task data harms scalability

Option to provide concise output

- Same basic format
- Omit collection of per callsite/task data

User controls output format through parameters

- -c = concise output only
- -v = provide concise and full output files

## Limiting Scope

---

By default, mpiP measures entire execution

- Any event between MPI\_Init and MPI\_Finalize

Optional: controlling mpiP from within the application

- Disable data collection at startup (-o)
- Enable using MPI\_Pcontrol(x)
  - x=0: Disable profiling
  - x=1: Enable profiling
  - x=2: Reset profile data
  - x=3: Generate full report
  - x=4: Generate concise report

## Limiting Scope / Example

```
for(i=1; i < 10; i++)  
{ switch(i)  
  {  
    case 5:  
      MPI_Pcontrol(2);  
      MPI_Pcontrol(1);  
      break;  
    case 6:  
      MPI_Pcontrol(0);  
      MPI_Pcontrol(4);  
      break;  
    default:  
      break; }  
  /* ... compute and communicate for one timestep ... */  
}
```

Reset & Start in Iteration 5

Stop & Report in Iteration 6

## mpiP Case Study

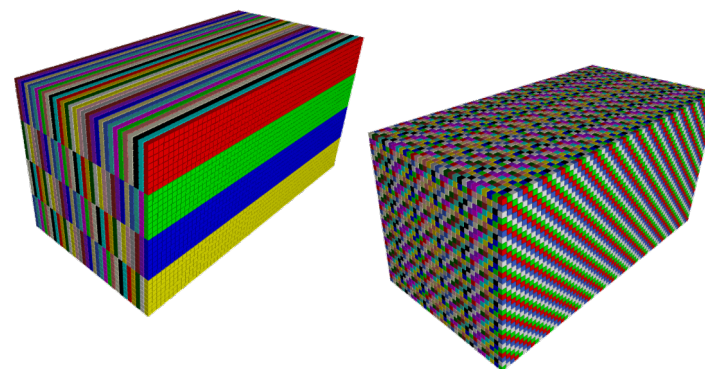
---

First Principle Molecular Dynamics Code @ LLNL

- Tuning at 65.536 nodes on BG/L

Two main observations

- Few MPI\_Bcasts dominate
  - Better node mappings
    - Up to 64% speedup
- Significant time spent in datatype operations
  - Hidden in underlying support libraries
    - Simplified data type management
      - 30% performance improvement



## Platforms

---

Highly portable design

- Built on top of PMPI, which is part of any MPI
- Very few dependencies

Tested on many platforms, including

- Linux (x86, x86-64, IA-64, MIPS64)
- BG/L & BG/P
- AIX (Power 3/4/5)
- Cray XT3/4/5 with Catamount and CNL
- Cray X1E

## mpiP Installation

---

Download from <https://github.com/LLNL/mpiP>

- Current release version: 3.5
- Git access to development version

Autoconf-based build system with options to

- Disable I/O support
- Pick a timing option
- Choose name demangling scheme
- Build on top of the suitable stack tracer
- Set maximal stack trace depth

OR: via Spack



# MPIP = Simple MPI Profiling

Quick overview of MPI traffic  
Simple Installation and inclusion  
Ability to tailor measurements

<https://github.com/LLNL/mipiP>