Why do we need another programing model ?

Atsushi Hori Min Si
Riken ANL

B. Gerofi, M. Takagi, Y. Ishikawa (RIKEN)
J. Dayal (Intel), P. Balaji (ANL)

HPDC'18 Main Conference

Thursday, 14 June

10:30 - 12:00 **Session 4 - Runtime Systems (Memorial Union Ventana B&C)**

PShifter: Feedback-based Dynamic Power Shifting within HPC Jobs for Performance
Neha Gholkar, Frank Mueller (North Carolina State University); Barry Rountree, Aniruddha Prakash Marathe (Lawrence Livermore National Laboratory)

ADAPT: An Event-based Adaptive Collective Communication Framework
Xi Luo (University of Tennessee, Knoxville); Wu Wei (Los Alamos National Laboratory); George Bosilca, Thananon Patinyasakdikul, Jack Dongarra (University of Tennessee, Knoxville); Linnan Wang (Brown University)

Process-in-Process: Techniques for Practical Address-Space Sharing
Atsushi Hori (RIKEN); Min Si (ANL); Balazs Gerofi, Masamichi Takagi (RIKEN); Jai Dayal (Intel);
Pavan Balaji (ANL); Yutaka Ishikawa (RIKEN)

Outline

- Multi-process and Multi-thread
  - Historical background
- Motivation
- New Execution Model
- Process-in-Process (PiP)
- Showing some numbers

Multi-Process

- Beginning
  - Multi-programming
    - Running “independent” programs at the same time
  - Multi-tasking and Time-sharing
    - Utilizing CPU idle time
- Nowadays (in HPC)
  - Running “familiar” programs
  - No need of utilizing idle CPU time (busy-wait)
  - Frequent communication among processes
    - IPC (e.g., pipes, sockets, …) is too heavy
    - Shared memory is better, but …

Multi-Thread

- Beginning
  - Interacting *Oversubscribed* Execution Entities
  - "Light-weight" process
    - Fast creation
      - Not loading and linking a program, but creating new context (incl. stack)
    - Easy to exchange information
- Nowadays
  - Its creation is still heavy
    - not to create threads on-demand
  - No *oversubscription*
  - Shared variables must be protected

My Experience

- A decade ago,
  developing low-level intra-node communication library for MPI
- By using shared mmap
  - Not easy at all !!
    - Setup part is NOT easy
    - Communication part is easy

- Wait, something is wrong
  - A process cannot access the other process
  - Processes access the same PHYSICAL memory !!
  - It is the OS to create the inter-process barrier

And Many-Core

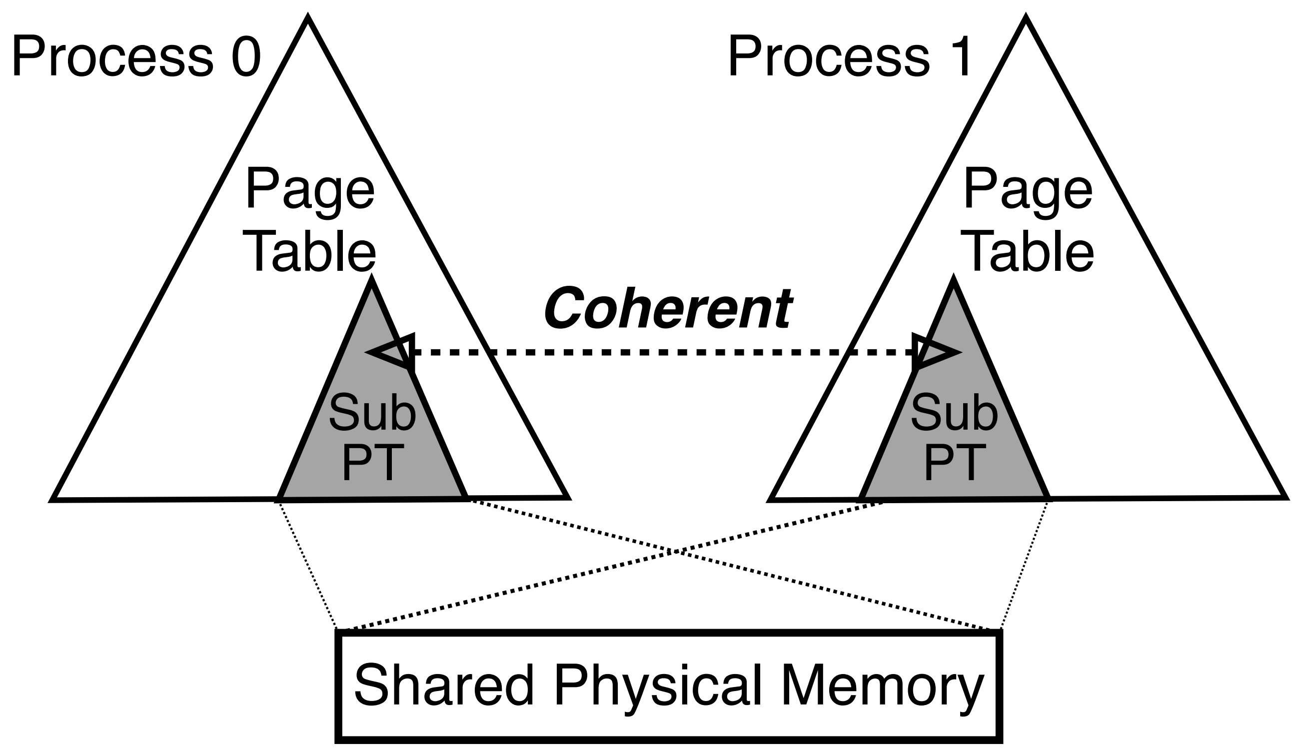
- More parallelism in a node
  - from 10^0 to 10^2 (or more)
- More interaction between processes or threads
  - Multi-Process: Hard to communicate
  - Multi-Thread: Shared variables must be protected

- We need something new (if you are not happy)
  - Easy to communicate
  - No shared variables

Shared Memory and XPMEM

- “Hole in the wall” to go through the barrier
  - Need of 2 copies to pass data
  - Pointers in the shared memory are useless

  - Setup (creation) cost
  - Need of page table entries to map
  - Coherency (page fault) overhead



Let's Break the Wall !

- Not making a tiny hole in the wall,
  but removing the whole wall !!!

- Removing the ***walls*** between processes
  - Keep variables private as in the same way of multi-process

  ➡ Easy to exchange data as easy as multi-thread because there is no ***wall***

AND

- Build another ***fence*** between threads
  - Make variables private to each thread

  ➡ No need of protection on shared variables

3rd Execution Model

		Address Space	
		Isolated	Shared
Variables	Privatized	Multi-Process (MPI)	***3rd Exec. Model***
	Shared	*N/A*	Multi-Thread (OpenMP)

Implementation

- This idea is not new
- Pack processes into one virtual address space
  - SMARTMAP (SNL)
  - PVAS (Riken)
- Threads pretending processes
  - MPC (CEA)
    - Need of special compiler to privatize variables, converting static variables to TLS variables

SMARTMAP and PVAS

Process 0
Process 1
:
Process n-1
Kernel

Make it more practical and portable

- No need of virtual address space partitioning
  - Only OS can partition virtual address space

- *Process-in-Process (PiP)*
  - User-level library
  - Implementation
    - dl**m**open() to privatize variables
    - create execution entities (processes or threads) to share the same virtual address space
      - i.e., clone() or pthread_create()
    - PiP programs must be PIE so that dlmopen() can load programs in different locations

/proc/*/maps example of PiP

```
555555554000-555555556000 r-xp ... /PIP/test/basic
555555755000-555555756000 r--p ... /PIP/test/basic
555555756000-555555757000 rw-p ... /PIP/test/basic
555555757000-555555778000 rw-p ...  [heap]
7fffe8000000-7fffe8021000 rw-p ...
7fffe8021000-7fffec000000 ---p ...
7ffff0000000-7ffff0021000 rw-p ...
7ffff0021000-7ffff4000000 ---p ...
7ffff4b24000-7ffff4c24000 rw-p ...
7ffff4c24000-7ffff4c27000 r-xp ... /PIP/lib/libpip.so
7ffff4c27000-7ffff4e26000 ---p ... /PIP/lib/libpip.so
7ffff4e26000-7ffff4e27000 r--p ... /PIP/lib/libpip.so
7ffff4e27000-7ffff4e28000 rw-p ... /PIP/lib/libpip.so
7ffff4e28000-7ffff4e2a000 r-xp ... /PIP/test/basic
7ffff4e2a000-7ffff5029000 ---p ... /PIP/test/basic
7ffff5029000-7ffff502a000 r--p ... /PIP/test/basic
7ffff502a000-7ffff502b000 rw-p ... /PIP/test/basic
7ffff502b000-7ffff502e000 r-xp ... /PIP/lib/libpip.so
7ffff502e000-7ffff522d000 ---p ... /PIP/lib/libpip.so
7ffff522d000-7ffff522e000 r--p ... /PIP/lib/libpip.so
7ffff522e000-7ffff522f000 rw-p ... /PIP/lib/libpip.so
7ffff522f000-7ffff5231000 r-xp ... /PIP/test/basic
7ffff5231000-7ffff5430000 ---p ... /PIP/test/basic
7ffff5430000-7ffff5431000 r--p ... /PIP/test/basic
7ffff5431000-7ffff5432000 rw-p ... /PIP/test/basic
...
7ffff5a52000-7ffff5a56000 rw-p ...
...
7ffff5c6e000-7ffff5c72000 rw-p ...
7ffff5c72000-7ffff5e28000 r-xp ... /lib64/libc.so
7ffff5e28000-7ffff6028000 ---p ... /lib64/libc.so
7ffff6028000-7ffff602c000 r--p ... /lib64/libc.so
7ffff602c000-7ffff602e000 rw-p ... /lib64/libc.so
```

Program

Glibc

```
7ffff602e000-7ffff6033000 rw-p ...
7ffff6033000-7ffff61e9000 r-xp ... /lib64/libc.so
7ffff61e9000-7ffff63e9000 ---p ... /lib64/libc.so
7ffff63e9000-7ffff63ed000 r--p ... /lib64/libc.so
7ffff63ed000-7ffff63ef000 rw-p ... /lib64/libc.so
7ffff63ef000-7ffff63f4000 rw-p ...
7ffff63f4000-7ffff63f5000 ---p ...
7ffff63f5000-7ffff6bf5000 rw-p ... [stack:10641]
7ffff6bf5000-7ffff6bf6000 ---p ...
7ffff6bf6000-7ffff73f6000 rw-p ... [stack:10640]
7ffff73f6000-7ffff75ac000 r-xp ... /lib64/libc.so
7ffff75ac000-7ffff77ac000 ---p ... /lib64/libc.so
7ffff77ac000-7ffff77b0000 r--p ... /lib64/libc.so
7ffff77b0000-7ffff77b2000 rw-p ... /lib64/libc.so
7ffff77b2000-7ffff77b7000 rw-p ...
...
7ffff79cf000-7ffff79d3000 rw-p ...
7ffff79d3000-7ffff79d6000 r-xp ... /PIP/lib/libpip.so
7ffff79d6000-7ffff7bd5000 ---p ... /PIP/lib/libpip.so
7ffff7bd5000-7ffff7bd6000 r--p ... /PIP/lib/libpip.so
7ffff7bd6000-7ffff7bd7000 rw-p ... /PIP/lib/libpip.so
7ffff7ddb000-7ffff7dfc000 r-xp ... /lib64/ld.so
7ffff7edc000-7ffff7fe0000 rw-p ...
7ffff7ff7000-7ffff7ffa000 rw-p ...
7ffff7ffa000-7ffff7ffc000 r-xp ... [vdso]
7ffff7ffc000-7ffff7ffd000 r--p ... /lib64/ld.so
7ffff7ffd000-7ffff7ffe000 rw-p ... /lib64/ld.so
7ffff7ffe000-7ffff7fff000 rw-p ...
7ffffffde000-7ffffffff000 rw-p ... [stack]
ffffffffff600000-ffffffffff601000 r-xp ... [vsyscall]
```

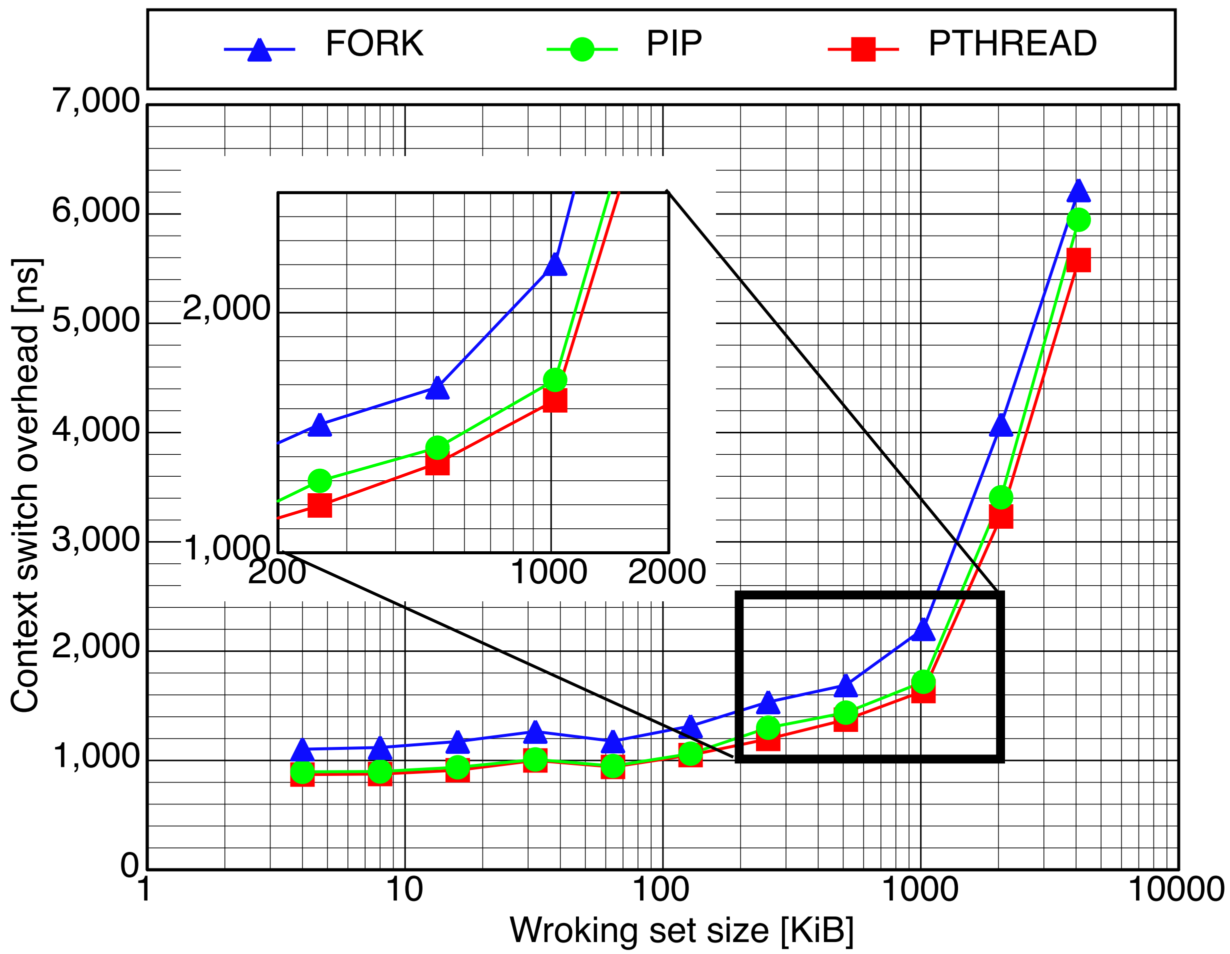
3rd Execution Model

		Address Space	
		Isolated	Shared
Variables	Privatized	Multi-Process (MPI)	***3rd Exec. Model***
	Shared	*N/A*	Multi-Thread (OpenMP)

Sharing a Page Table

- Do PiP tasks and the root share the same page table ?
  - Evaluation of switching two tasks using futex

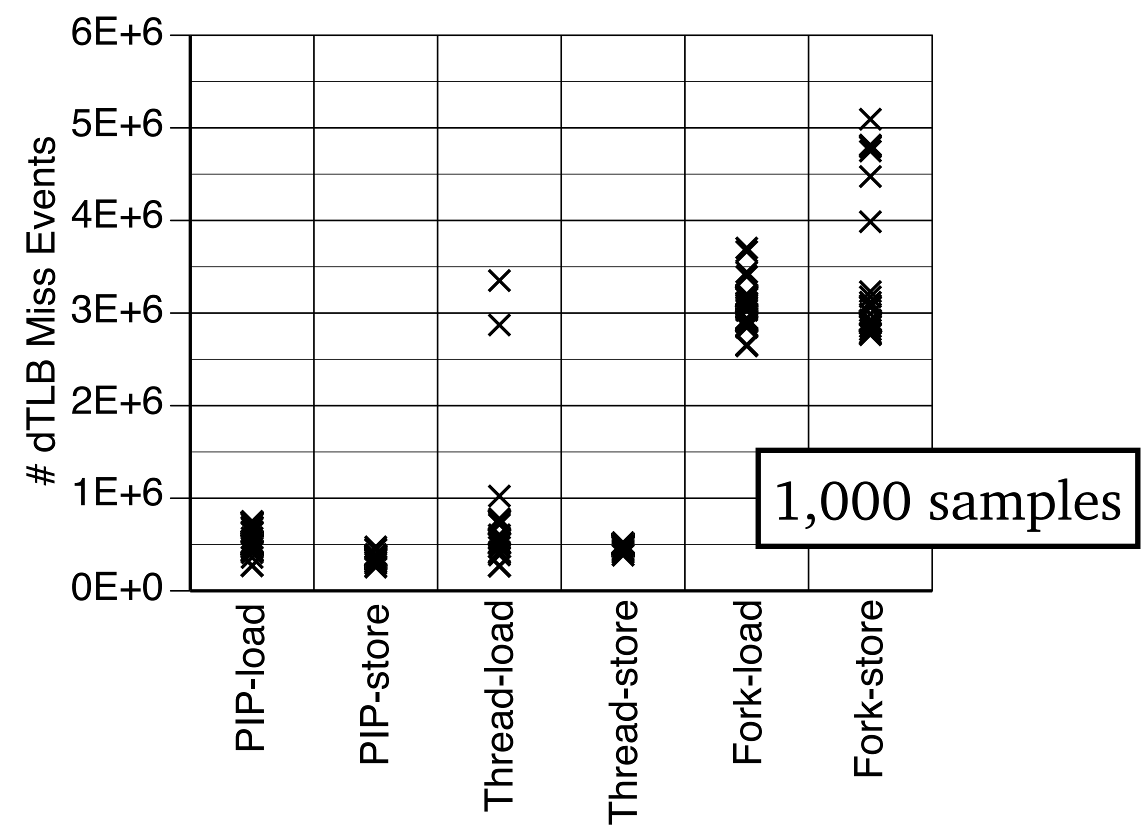
B. Sigoure. How long does it take to make a context switch?, November 2010.
http://blog.tsunanet.net/2010/11/how-long-does-it-take-to-make-context.html



Xeon E5-2650 v2 8×2(×2) 2.6GHz 64 GiB

Number of `load_cr3` function calls

PIP	Pthread	Fork
74.1	53.0	794535.4



How PiP works

- Execution Model
  - PiP Root Process
    - Root can spawn PiP tasks in the same virtual address space of the root
  - PiP Tasks
    - spawned by the root
- Execution Mode
  - Process mode
    - Tasks are created by clone()
  - Thread mode
    - Tasks are created by pthread_create()
    - Variables are privatized though

PiP vs. Shared Memory

- Setup Cost
- Page Table Size
- Number of Page Faults

Setup Cost

Allocating 2 GiB Shared Memory

XPMEM	Cycles
`xpmem_make()`	1,585
`xpmem_get()`	15,294
`xpmem_attach()`	2,414
`xpmem_detach()`	19,183
`xpmem_release()`	693

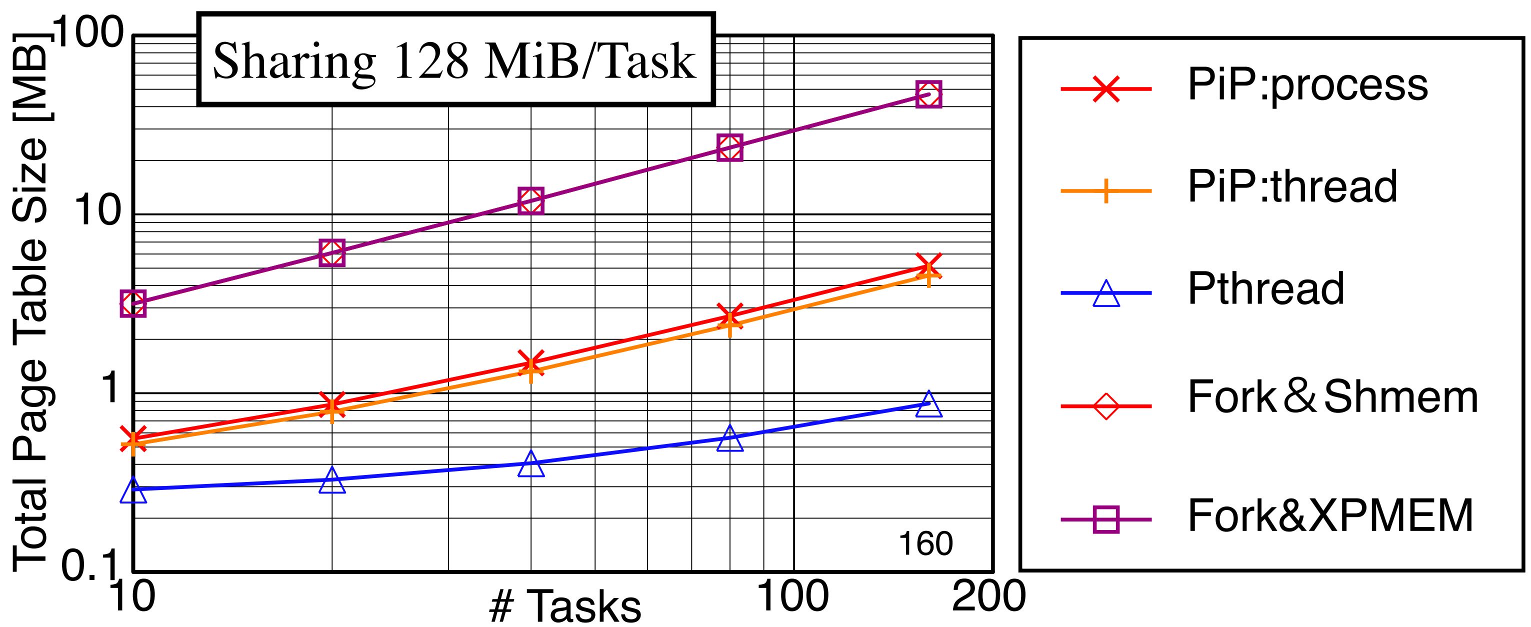
	POSIX Shmem	Cycles
Sender	`shm_open()`	22,294
	`ftruncate()`	4,080
	`mmap()`	5,553
	`close()`	6,017
Receiver	`shm_open()`	13,522
	`mmap()`	16,232
	`close()`	16,746

<pip/xpmem.h>

```
xpmem_segid_t xpmem_make( void *vaddr, size_t size,
                          int permit_type, void *permit_value ) {
  return (xpmem_segid_t) vaddr; }

int xpmem_remove( xpmem_segid_t segid ) { return 0;}

xpmem_apid_t xpmem_get( xpmem_segid_t segid,
                        int flags, int permit_type,
                        void *permit_value ) {
  return segid; }

int xpmem_release( xpmem_apid_t apid ) { return 0; }

void *xpmem_attach( struct xpmem_addr addr,
                    size_t size, void *vaddr ) {
  return (void*) ( addr.apid + addr.offset ); }

int xpmem_detach( void *vaddr ) { return 0; }
```

Xeon E5-2650 v2 8×2(×2) 2.6GHz 64 GiB

Page Table Size



Note: The results of Fork&Shmem and Fork&XPMEM are overlapped.

Figure 6: Total page table size running on Wallaby/Linux

Table 5: Total number of page table entries

	Total Number of Page Table Entries
Pthread	$M + D + \sum S_i$
PiP	$M + \sum D_i + \sum S_i$
Process + POSIX shmem	$(M \times N) + \sum D_i + \sum S_i$
Process + XPMEM	$(M \times N) + \sum D_i + \sum S_i$

M is the number of PT entries for the shared-memory region(s).
S_i is the number of PT entries for the stack segment of task i.
D_i is the number of PT entries to map shared objects belonging to task i.
N is the number of tasks (processes or threads).

Xeon E5-2650 v2 8×2(×2) 2.6GHz 64 GiB

Page Fault

- Sender allocates memory region and set some values
- Receiver scan the data in the "shared" memory region
  - Measure the each access time on the reciever

Xeon E5-2650 v2 8×2(×2) 2.6GHz 64 GiB

PiP Applications

- PiP application performance numbers will be shown in the main conference talk

- MPI
  - pt2pt communication
  - MPI_Win_alloate_shared()
- In-situ
  - By putting simulation program and in-situ program in the same virtual address space
  - 2 memory copies can be avoided
- MPI+OpenMP vs. MPI+PiP

Myths on PiP

- It is crazy to mix programs, I cannot debug !
  - Can't you debug multi-thread programs ?
- By using huge pages, PiP has no advantage !
  - PiP can work with huge pages
  - Pit falls of using huge pages
    - Transparent Huge Pages may hinder execution
    - Other Huge Page techniques need extra programming
    - Consumes more memory
- Shared memory is enough
  - PiP can do better than shared memory

PiP Summary

- 3rd parallel execution model
- User-level Implementation
  - No partitioning of virtual address space
- dlmopen(), PIE, and clone()
- Load multi-programs into the same virtual address space
- No communication (≈ copy),
  but accessing (no copy) by sharing virtual address space

Comparison

	Multi-Process	Multi-Thread	3rd Execution Model
Parallel Execution	yes	yes	yes
Sharing	nothing shared	VAS and variables	VAS
Execution starts	main	arbitrary func	main
Multi-programming	yes	no	yes

	SMARTMAP and PVAS	MPC	PiP
VAS sharing	yes	yes	yes
Based on	process	thread	process or thread
Multi-programming	yes	no	yes
Implementation	Kernel	Language	Library
Execution starts	main	any func	any func