High Performance Computing Systems '97

THREADS-ONLY MPI IMPLEMENTATION FOR DEVELOPMENT OF PARALLEL PROGRAMS ON A WORKSTATION

Erik Demaine
eddemaine@daisy.uwaterloo.ca

http://daisy.uwaterloo.ca/~eddemain/papers/HPCS97

July 10, 1997

OUTLINE

- Motivation
  - MPI
  - Problems with existing implementations
- TOMPI: exploiting uniprocessor environment
- Results
  - Comparisons to other implementations
- Conclusion

MOTIVATION: PARALLEL-PROGRAM DEVELOPMENT

- Want to develop parallel programs on a workstation
- Parallel computers
  - Expensive and hard to access
  - Impractical to share
- **Pseudo-parallel** environment simulating parallel one
- Want portability
  - Programs developed on workstation must port easily to target parallel computer
  - Existing parallel programs should run on workstation

MPI (MESSAGE PASSING INTERFACE)

- MPI: Popular standard for message passing
  - Implemented on most (distributed-memory) parallel machines
- To develop parallel programs on workstation, simply need MPI implementation for pseudo-parallel environment
- Several implementations for networks of UNIX workstations
  - CHIMP/MPI (EPCC)
  - LAM (OSC)
  - MPICH (ANL/MSU)
- All based on UNIX processes, which can multitask on single processor

PROBLEMS WITH EXISTING IMPLEMENTATIONS

- Recall target is parallel computer, potentially with
  - Fast network $\Longrightarrow$ fine-grain parallelism
  - Many processors $\Longrightarrow$ many communicating processes
- UNIX processes are expensive
  - High task switching and memory overhead
  - Alternative: threads
    - □ Share most of their state (including address space)
- Sockets are slow
  - Alternative: shared-memory communication
- Current implementations use polling
  - Alternative: locks and condition variables
    - □ Efficient at thread level

TOMPI: THREADS-ONLY MPI IMPLEMENTATION

- Only supports pseudo-parallel environment
  - Threads
  - Shared-memory communication
  - Blocking communication
- Queue structure for exchanging control messages
  - Constant-time insert/delete in arbitrary locations
  - Allocates by blocks
  - Incremental search
- Standard MPI send is synchronous
  - Common address space, so original message can be accessed directly
  - Threads offer fast synchronization

PREPROCESSOR

- Common address space $\Longrightarrow$ threads share global variables
- Need to use **thread-specific data** (TSD)
  - Create TSD "keys" shared by all threads
  - Thread accesses its version of data through function calls
- Preprocessor converts
  - Global-variable declarations to key declarations
  - References to function calls
- Uses little knowledge of C/C++
  - Lexical structure
  - Declarations
- Examples:

```
int i = 5;         ⟶  const int i = 5;
                      Key i_key = NULL;

i++;               ⟶  (*((int *) get_global (&i_key,
                          &i, sizeof (i))))++;

const int j = 5;   ⟶  const int j = 5;
```

RESULTS

- MPI implementations:

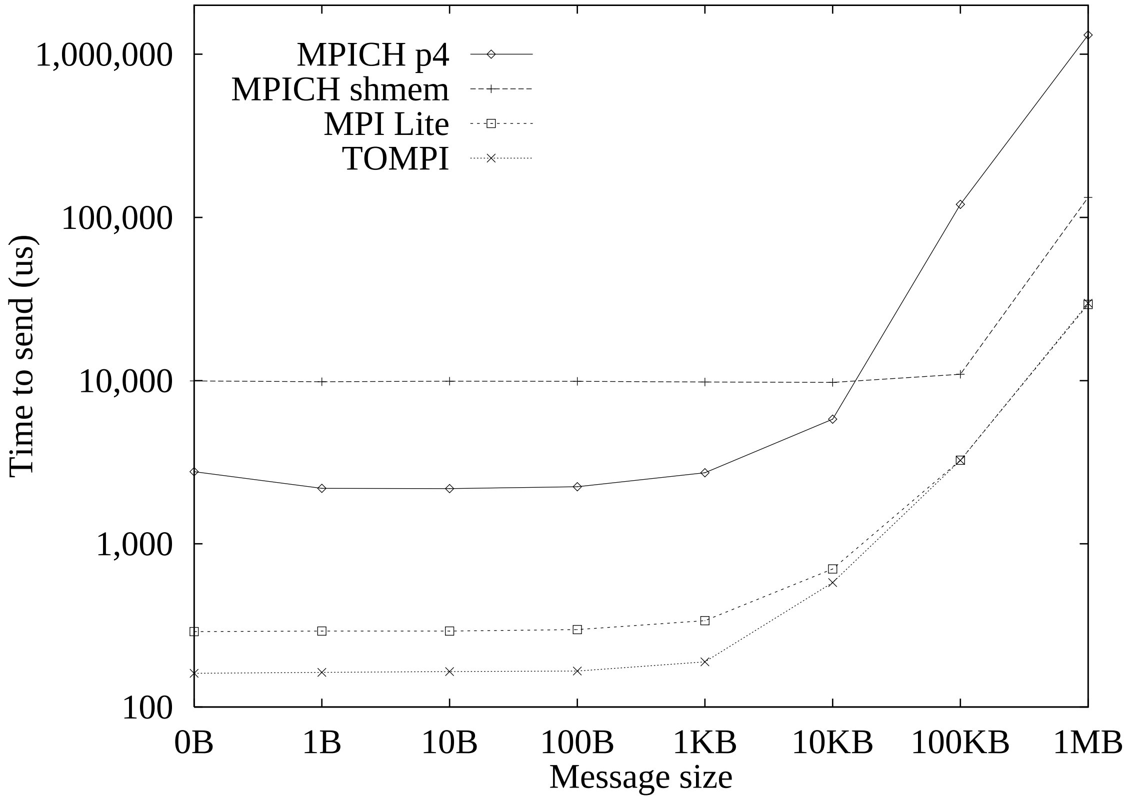
Implementation	Package	MPI Process	Communication	Polling
MPICH p4	UNIX	Process	Sockets	Yes
MPICH shmem	UNIX	Process	Shared memory	Yes
MPI Lite (UCLA)	Hand-written	Thread	Shared memory	No
TOMPI	POSIX/Solaris	Thread	Shared memory	No

- Machines:

Maker	Machine	Model	MHz	OS	Memory
IBM	RS/6000	25T	66	AIX 4.2.0	64
Sun	Ultra	1/170E	166	SunOS 5.5.1	128
Sun	SPARC SMP	10	2×75	SunOS 5.4	128

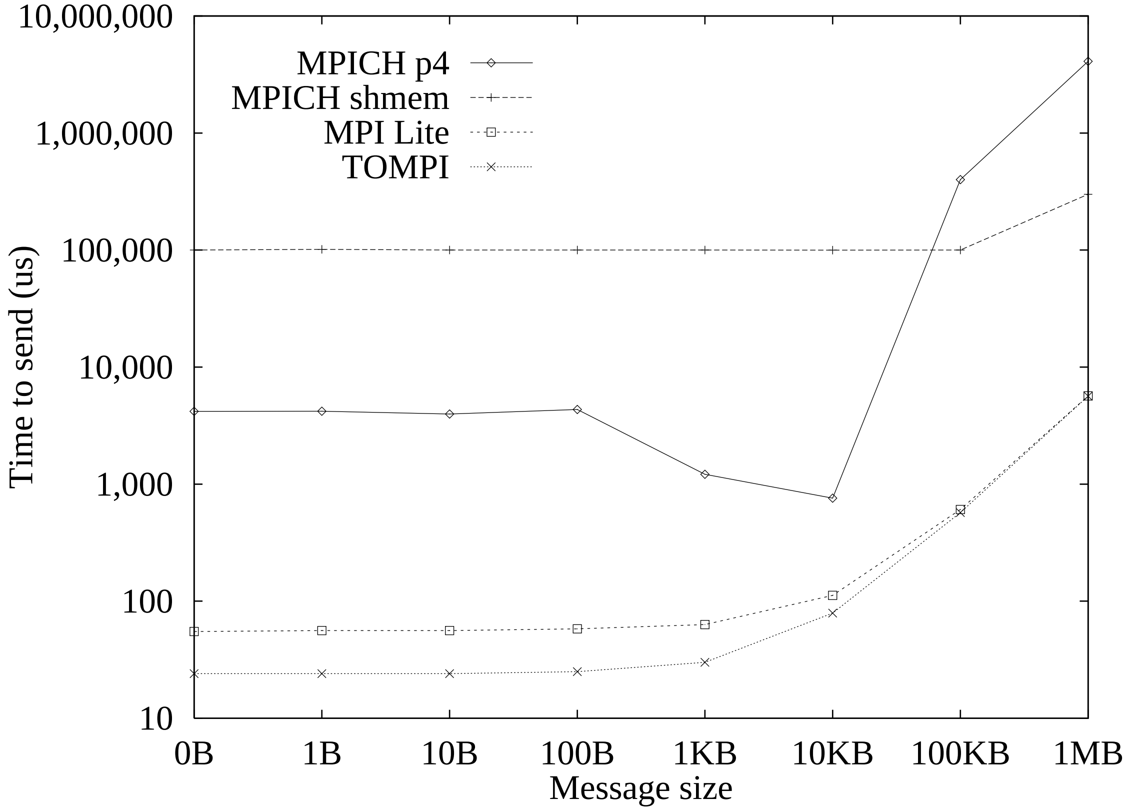
- Tests:
  - Ping-pong (communication speed)
  - Cholesky factorization

PING-PONG ON RS/6000



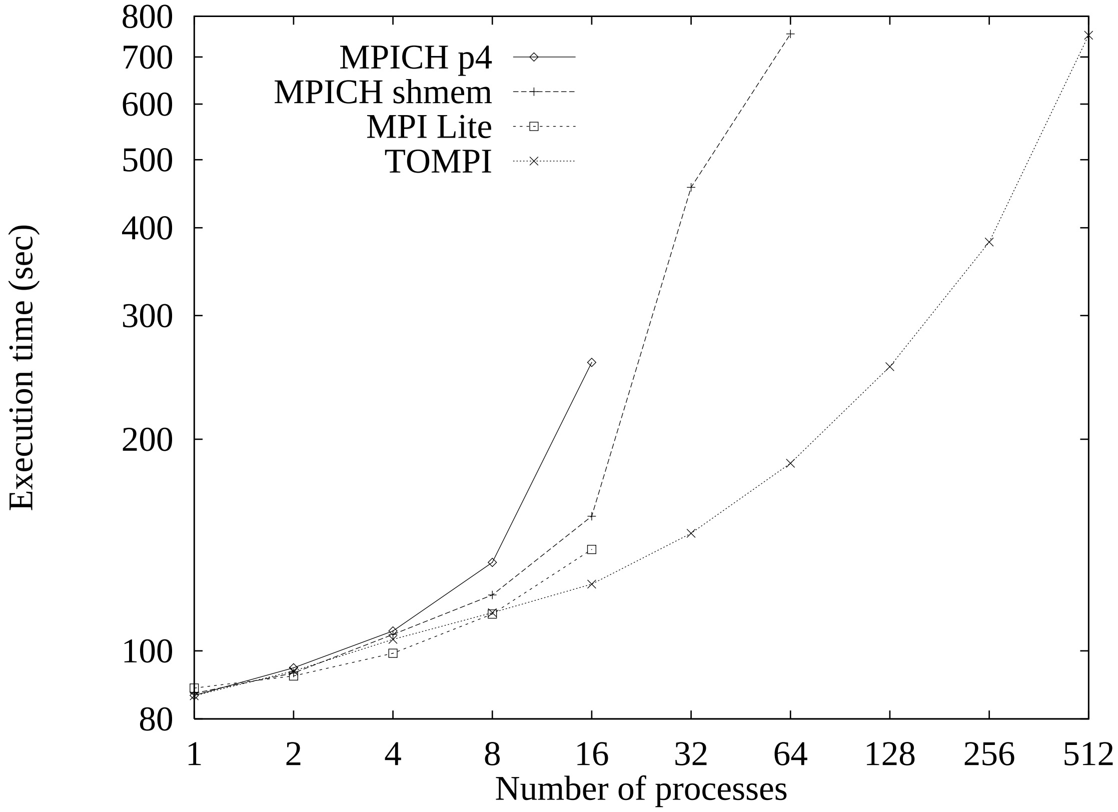
- MPICH
  - p4: sockets (slower than shared memory)
  - shmem: polls on variable (finer grain than socket)
  - For small messages (mostly polling), p4 wins
  - For large messages (mostly communication), shmem wins
- TOMPI is around twice as fast as MPI Lite for small messages
- Similar for large messages

PING-PONG ON ULTRA



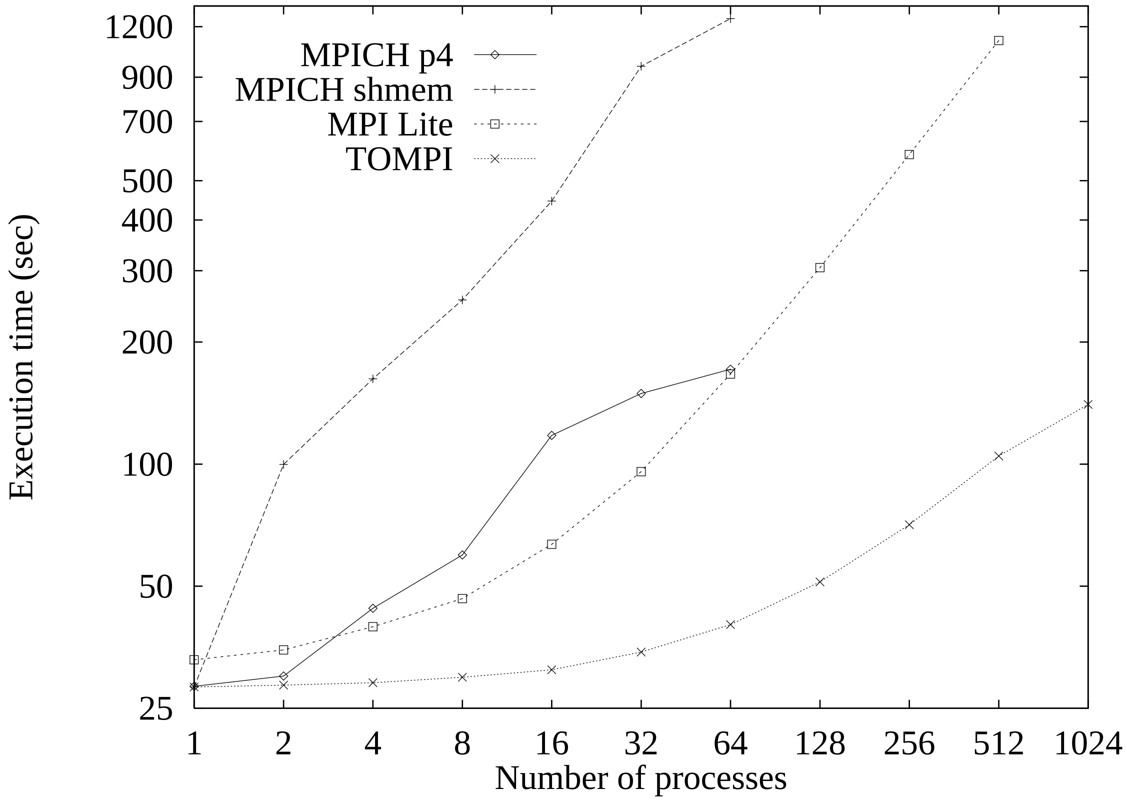
- MPICH
  - p4 is best when communication matches time-slice
  - Polling is more extreme (faster machine)
- Otherwise the same as RS/6000

CHOLESKY ON RS/6000



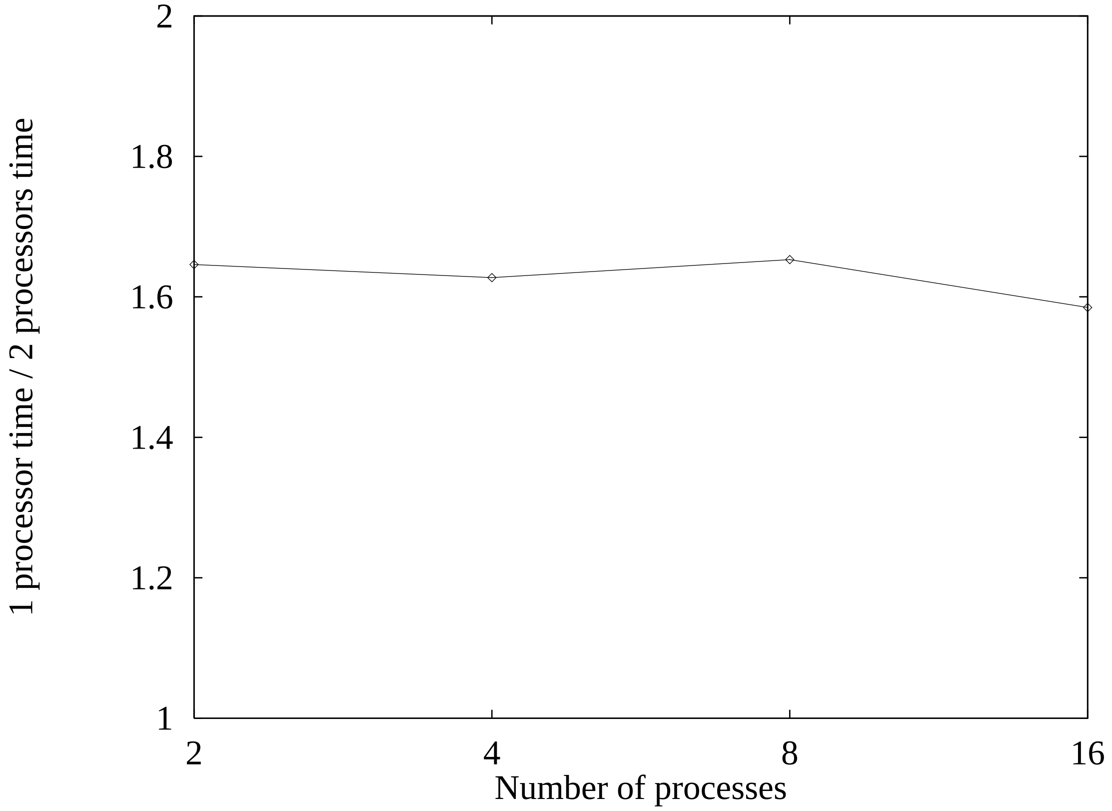
- All implementations are close for few processes
- MPICH: shmem beats p4 (mostly communication)
- MPI Lite: segmentation fault past 16 processes
- Only TOMPI makes it to 512 processes
  - AIX (system-level) threads enforce this limit
- TOMPI scales well

CHOLESKY ON ULTRA



- MPICH: shmem much worse (mostly polling)
- p4 is close to MPI Lite
- For 512 processes, TOMPI is 10$\times$ better
- Solaris threads allow 1024 processes and many more
- Slowdown under $10 = \log_2 1024$

CHOLESKY ON SMP



- Reasonable speedup (1.6) on SMP
- Buffered communication should improve this

CONCLUSION

- TOMPI: Develop parallel programs in pseudo-parallel environment
- Threads offer cheap parallelism
  - Requires preprocessor
- Lightweight communication
  - No polling
  - Synchronous communication by default
  - Queue structure

FUTURE WORK

- Efficiently support all of MPI
  - Collective communication
  - Fortran support
- Incorporate into existing MPI implementation (e.g., MPICH)
  - Similar to idea of MPI Lite
- Support more threads packages (especially user-level)