



Process-Based SMP Runtime

Filip Blagojević, Paul Hargrove,
Costin Iancu and Kathy Yelick

Outline

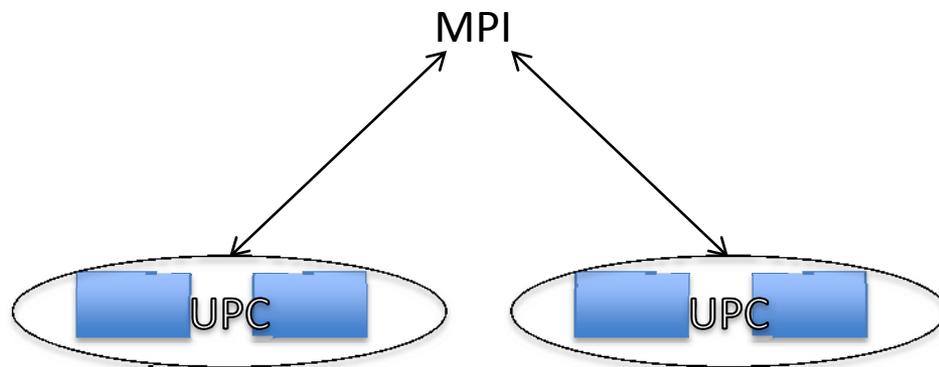


- Introduction
- Technical overview
- Evaluation on conventional CPUs
- Evaluation on accelerators
- Conclusions and Future Work

Introduction

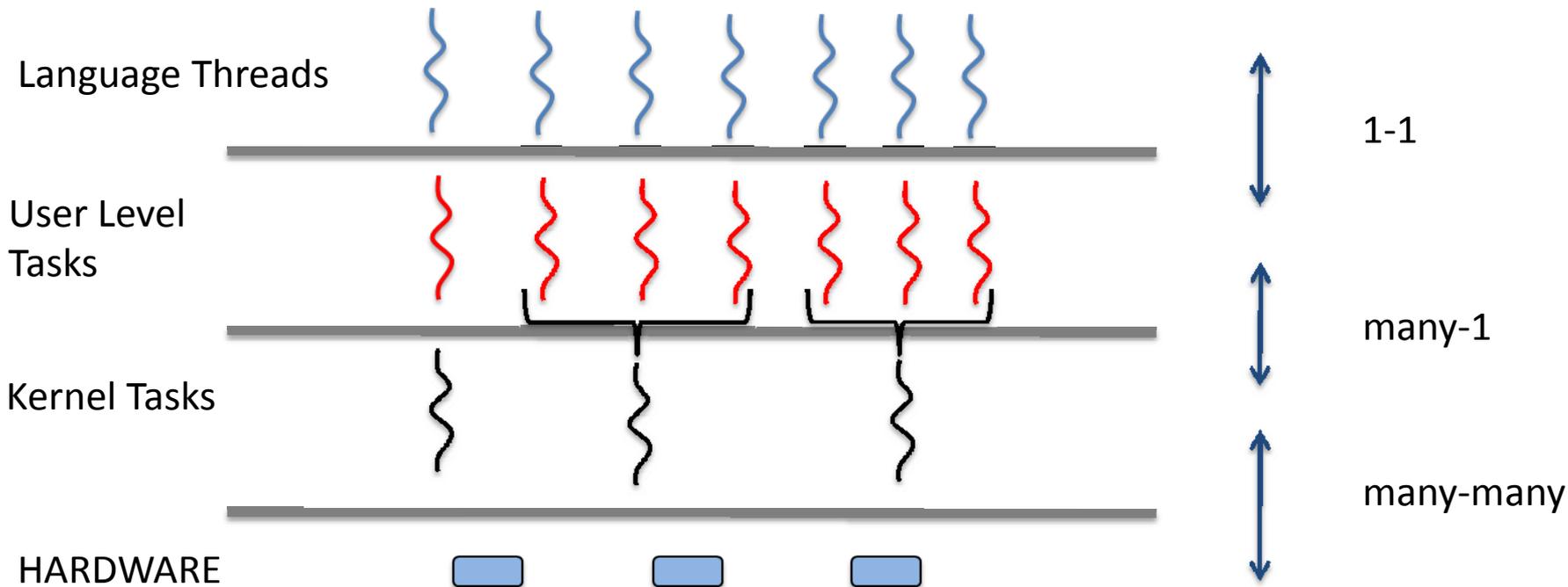


- Our goal:
 - Improve performance of UPC on multi-core shared memory machines
 - Improve interoperability (Hybrid Execution: MPI, OMP, UPC)

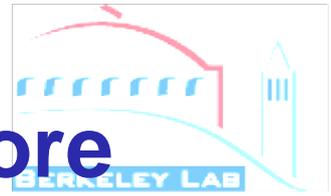


- Our work:
 - Investigate mapping of the Language threads to the OS threads
 - We implemented Process with Shared Memory (ProcSM) execution model in the UPC runtime
- Focus of the talk:
 - UPC execution on top of Pthreads
 - UPC execution on top of Processes (using ProcSM)

Introduction



- PGAS execution model:
 - PGAS Language Threads \leftrightarrow User Threads (or Processes)
 - User Threads \leftrightarrow Kernel Tasks
 - Kernel Tasks \leftrightarrow Hardware Execution Contexts



Threads VS Processes on Multicore

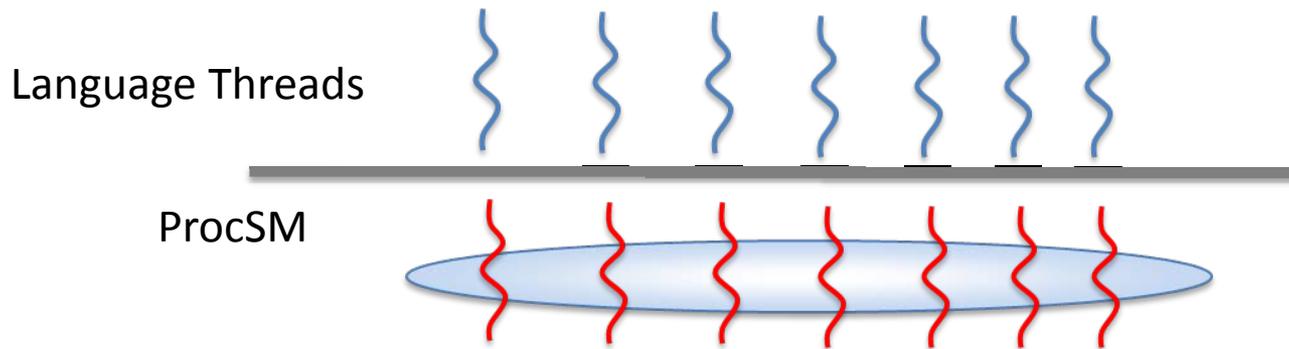
- Language threads map to threads or processes ?
- Threads:
 - Share address space
 - Easy and fast communication
 - Lighter than processes (in terms of context switching)
- Processes:
 - Require Kernel mechanisms or external libs for communication
 - Heavy Context Switch
 - Do not share TLB



UPC: Threads VS Processes

- UPC Threads map to Pthreads or processes
- UPC → Pthreads recommended on shared memory nodes
 - Communication performed through shared memory (processes use loopback)
- Pthreads/Processes Performance
 - Infiniband hardware allows 1 connection per process
 - Scheduling and CPU/Memory affinity?
 - Thread local data
- Pthreads disadvantage: Interoperability
 - Many libraries are not thread safe (Example: FFTW, C++ stl not thread safe, C I/O functions on certain OSs)
 - Hybrid Execution

Pthreads Alternative: Processes + Shared Memory

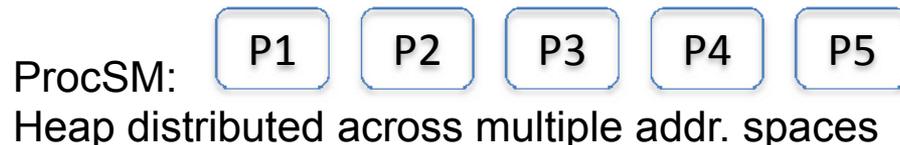
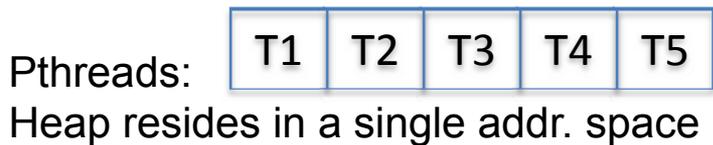


- Use ProcSM to improve interoperability
- Increase performance
- ProcSM implementation currently supported for SMPs and cluster execution with Infiniband and MPI networks

ProcSM UPC implementation

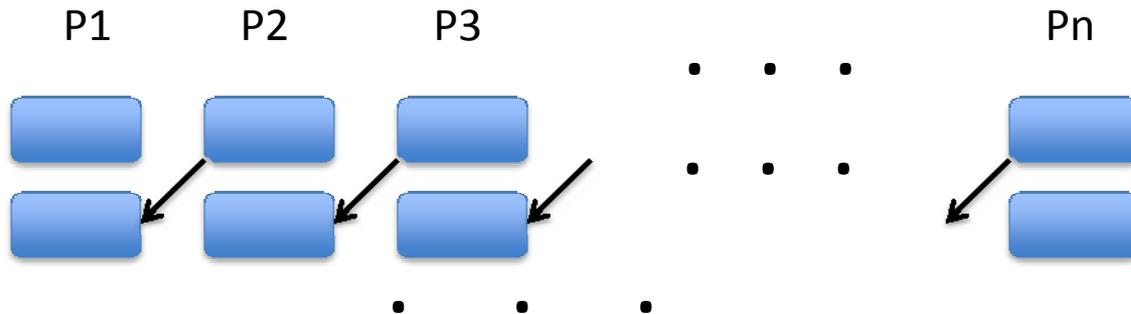


- Pthreads, ProcSM – Heap Implementation



- Implementation details

- Total of N+1 shared segments required
- ProcSM network segment is allocated and attached to all processes
- All threads allocate one memory segment; size is user controlled
- Each thread mmmaps segments of all other threads

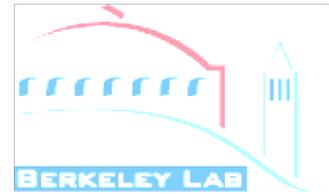


ProcSM - UPC implementation



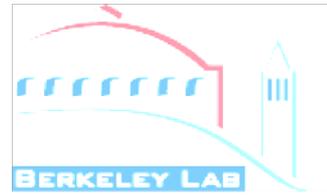
- Proc SM shared segment mapping
 - POSIX or SYSV
 - Anonymous mapping not possible with SYSV
 - POSIX SM does not create a file on disk – data written to a shared memory region is not disk-synchronized
- Shared memory network (SMNet)
 - Resides in a single SM region attached to all processes
 - Active Messages sent through the SMNet
 - Handler parameters exchanged directly among shared memory segments
- Shared pointers on the same physical node are accessed directly, without involving the network

Evaluation

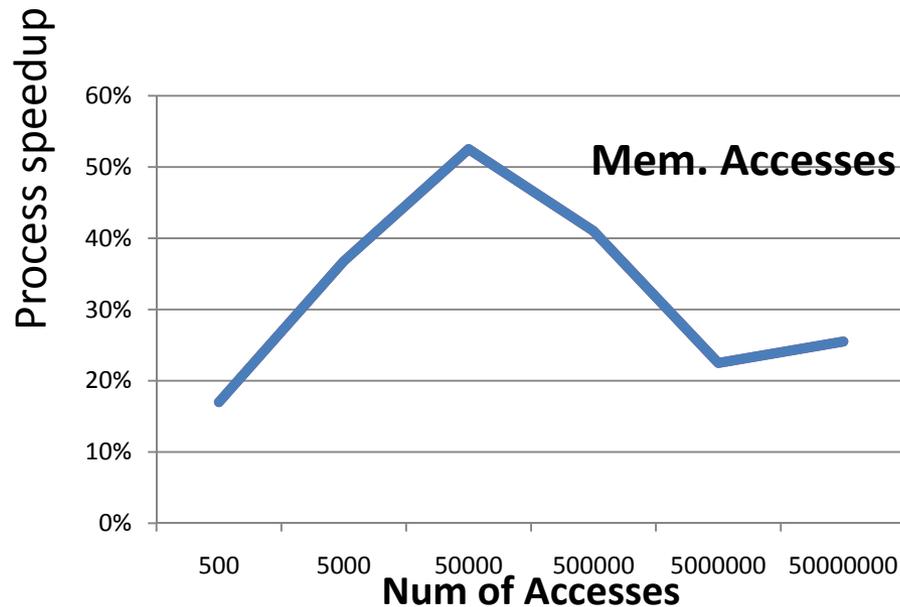


- AMD Barcelona
 - 4 Sockets – 4 core, NUMA
- Intel Tigerton
 - 4 Sockets – 4 core, UMA
 - 2 Nodes, Infiniband connection
- Ranger
 - Up to 64 cores
- Microbenchmarks
 - Shared memory access
 - Communications
- NAS Benchmarks
 - UPC Implementation
 - CG, EP, MG, IS, BT, SP, FT

Shared Memory Access

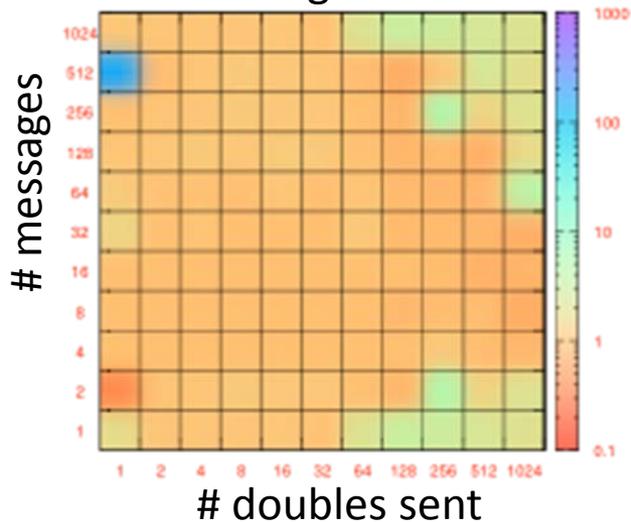


- AMD Barcelona
 - 4 Socket, 4 Cores
- Microbenchmark:
 - Shared global memory access – streaming
- Num of threads: 16



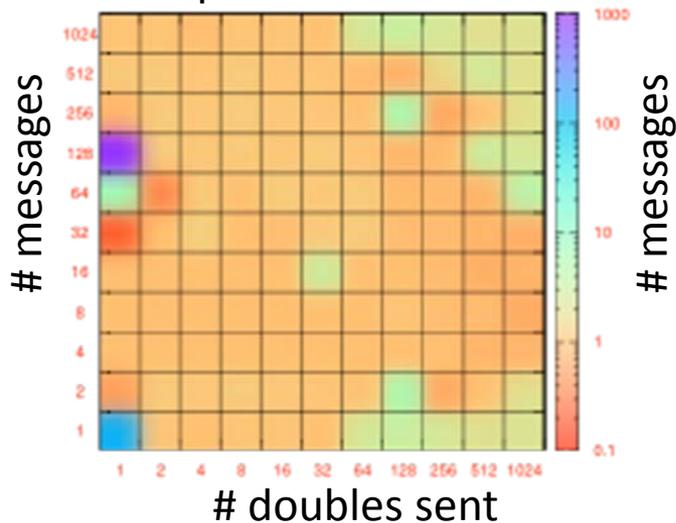
Communication Performance – Ranger 16 Cores

Blocking Comm.

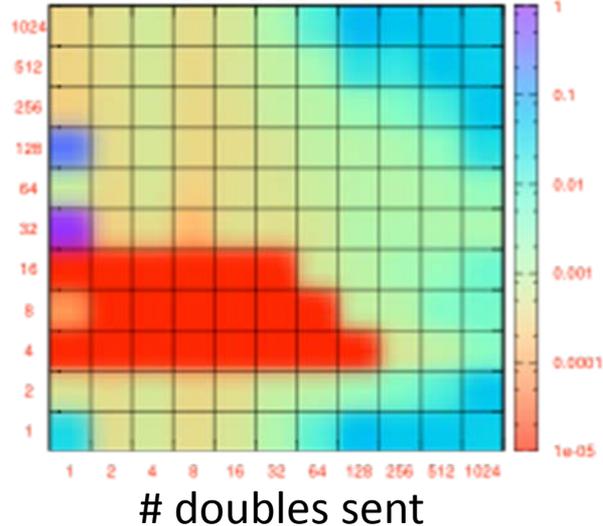


- Blocking and Pipelined comm. use rdma()
- Strided comm. use active messages
- For Blocking and Pipelined comm. performance similar
- ProcSM behavior more predictable

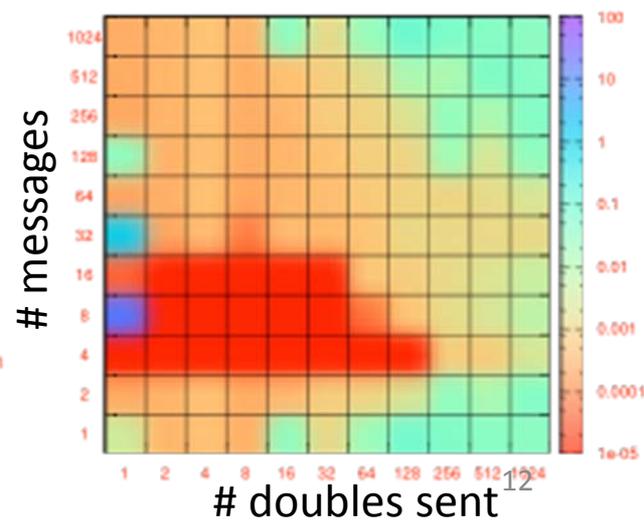
Pipelined Comm.



1D Strided Comm.

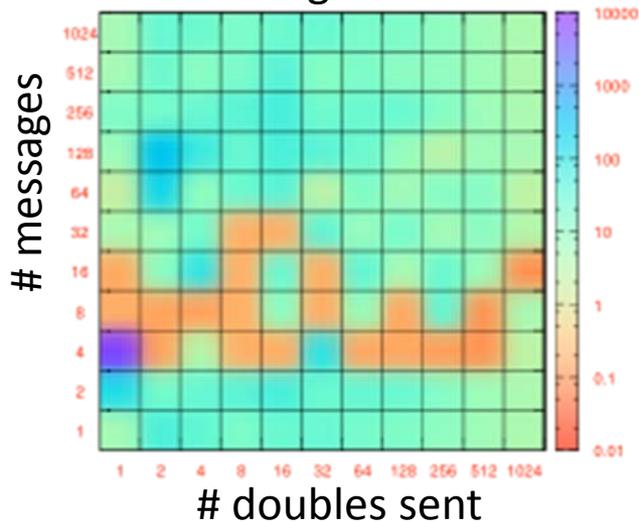


2D Strided Comm.



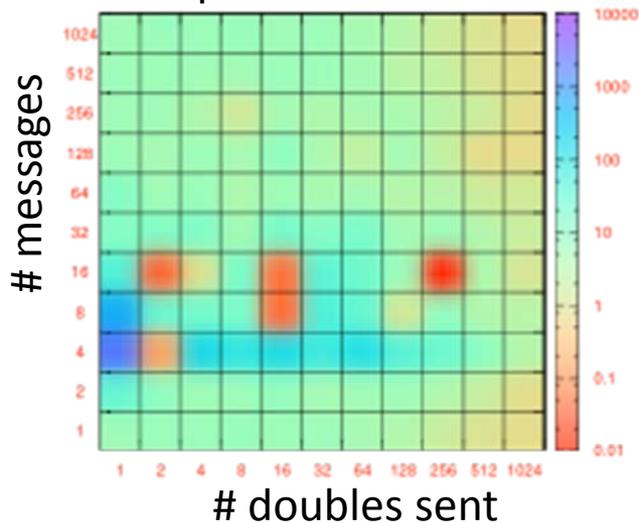
Communication Performance – Ranger 64 Cores

Blocking Comm.

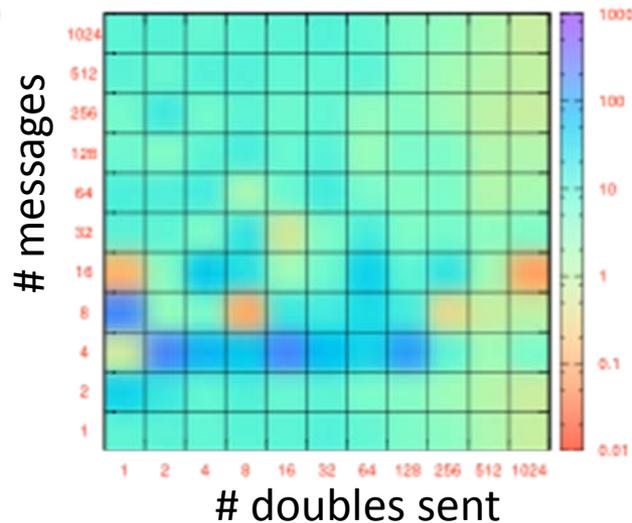


- ProcSM perform significantly better

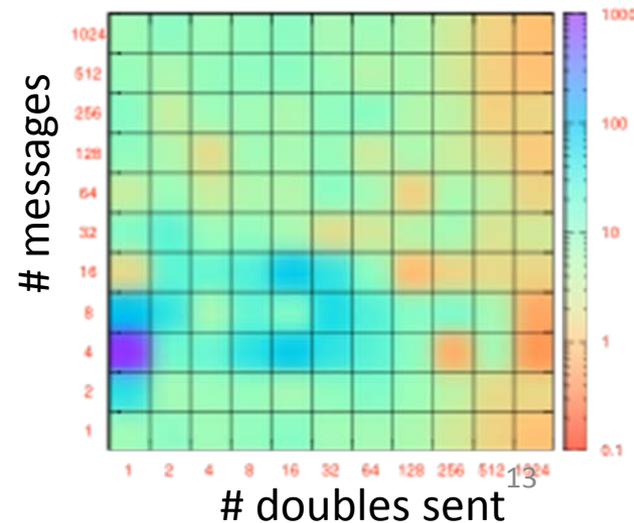
Pipelined Comm.



1D Strided Comm.



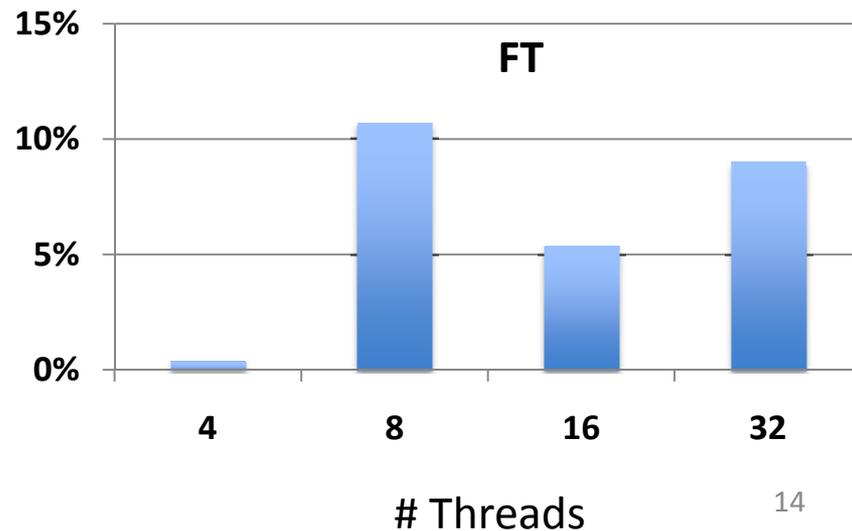
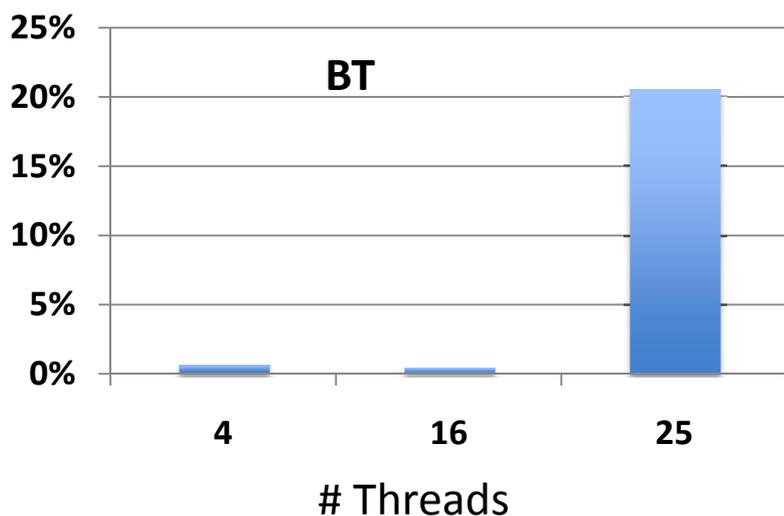
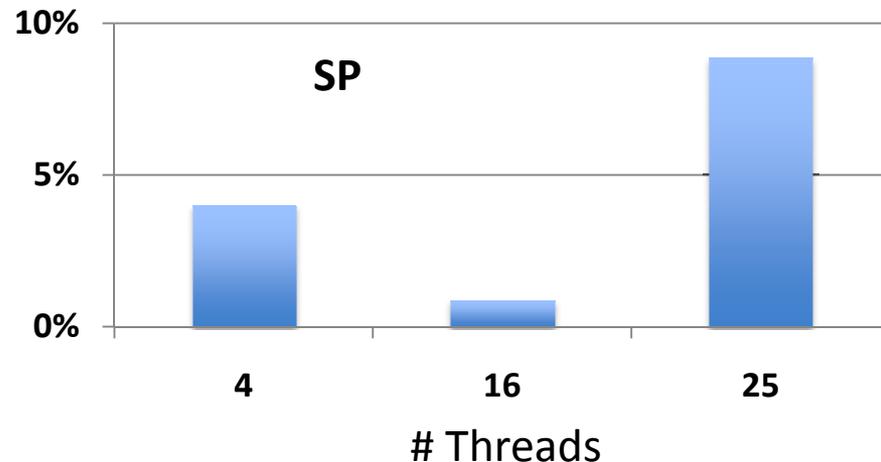
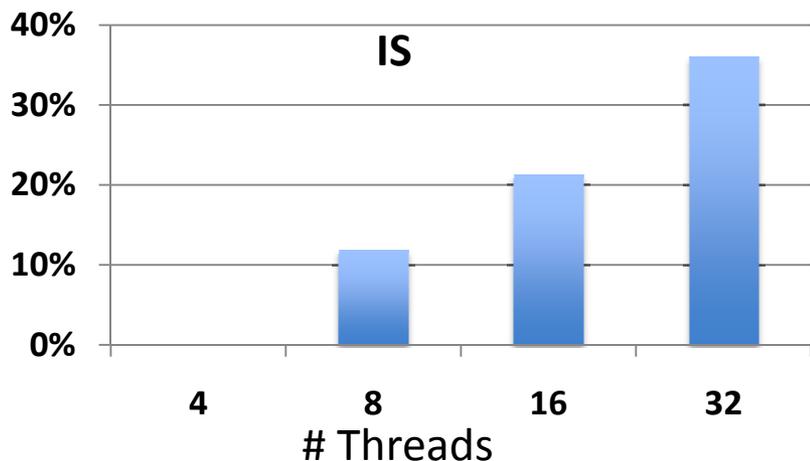
2D Strided Comm.





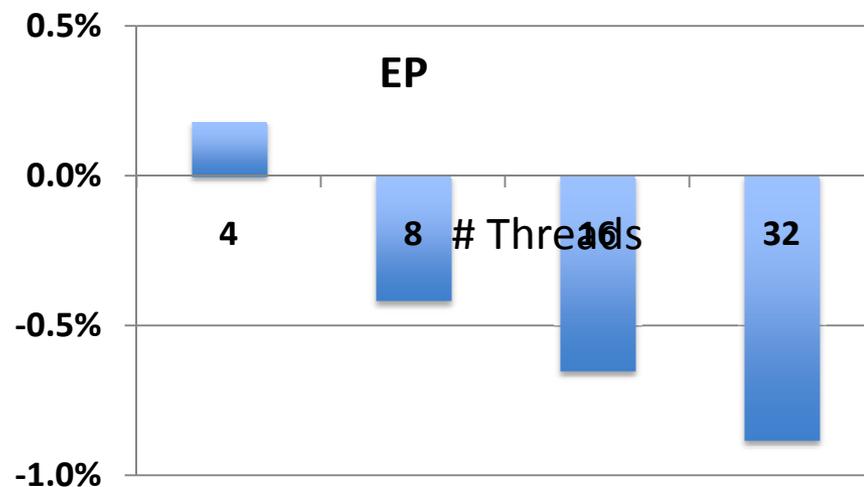
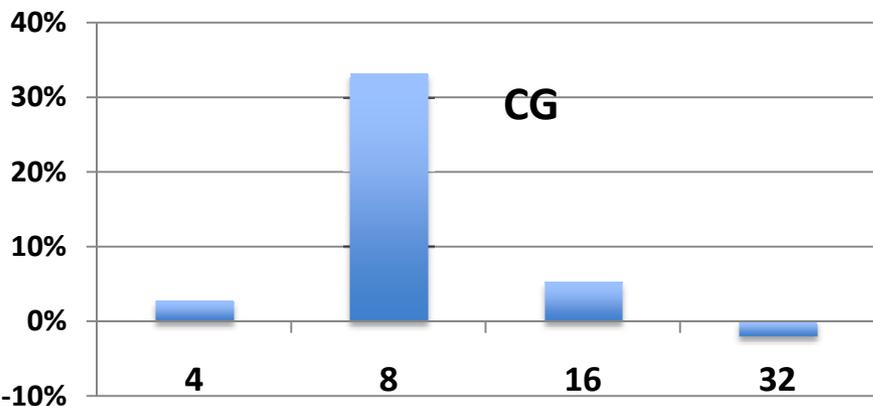
NAS Benchmarks – Intel Tigerton

Performance improvement of ProcSM over Pthreads

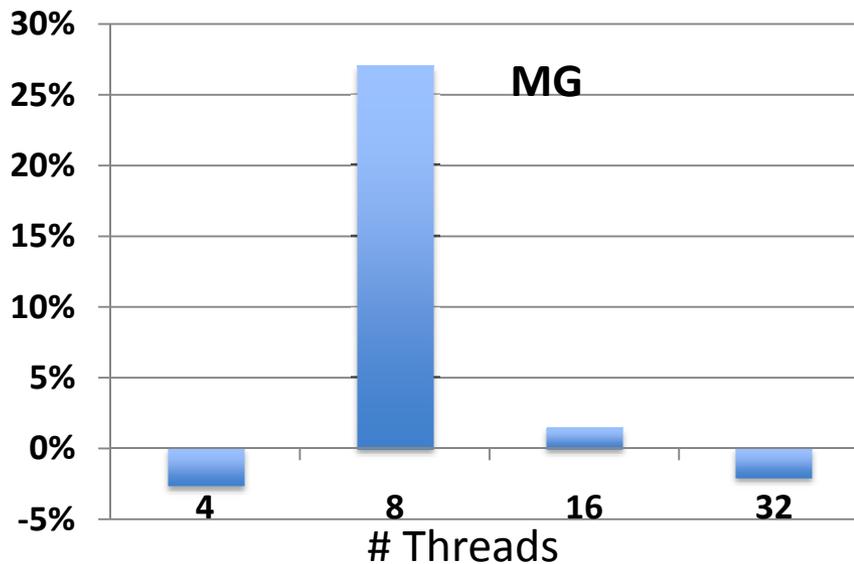


NAS Benchmarks – Intel Tigerton

Performance improvement of ProcSM over Pthreads



Threads

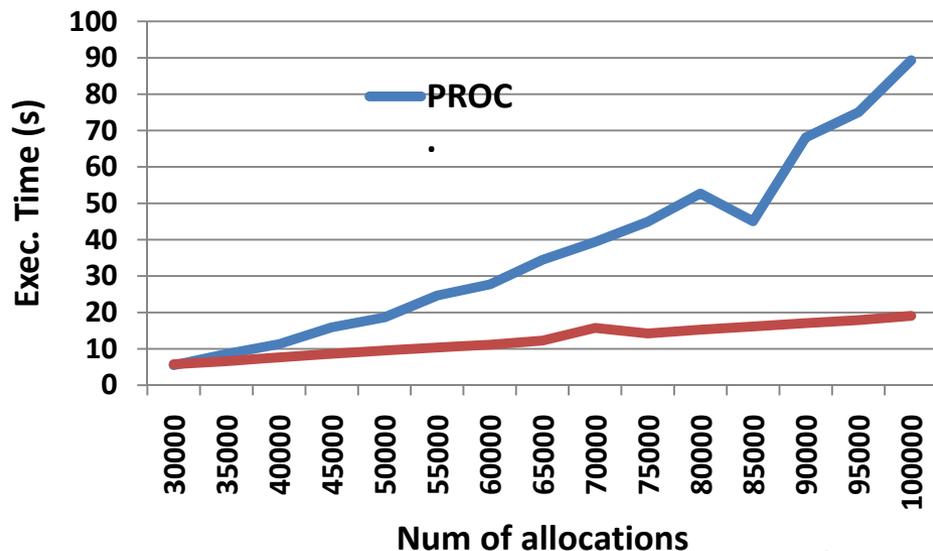
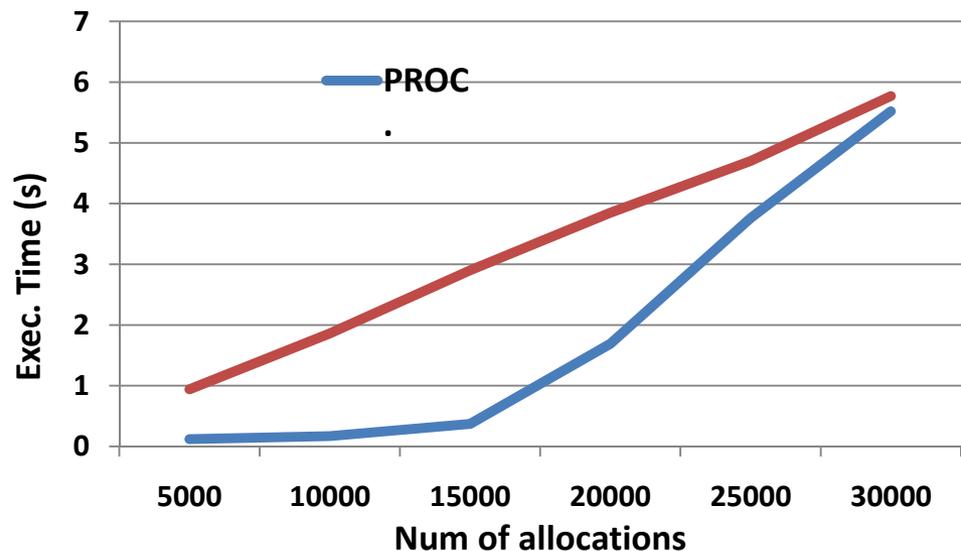




SYSV Speedup on multicore

- Possible reasons for performance improvement:
 - Thread Local data
 - Memory Allocation
 - Infiniband Hardware
 - ...

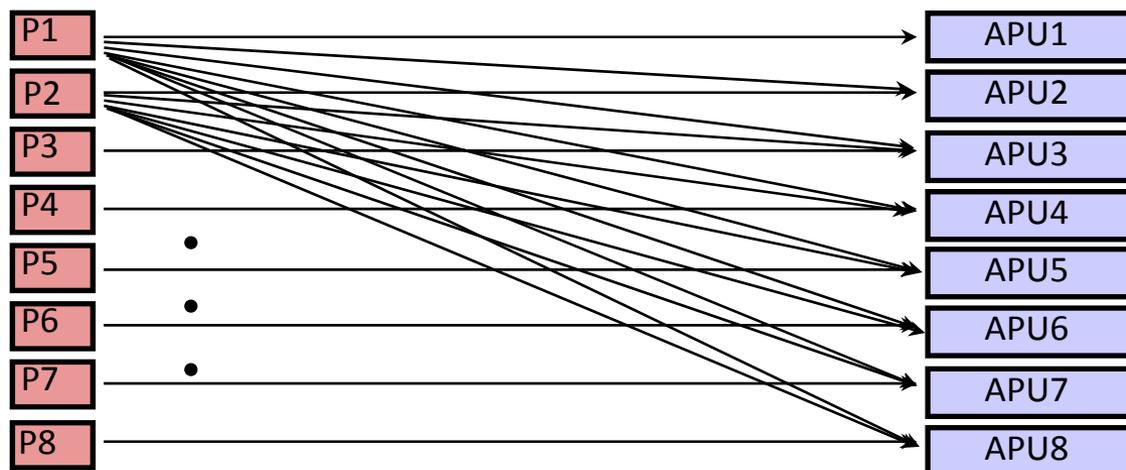
UPC_all_alloc() called in a large loop



ProcSM on Accelerators



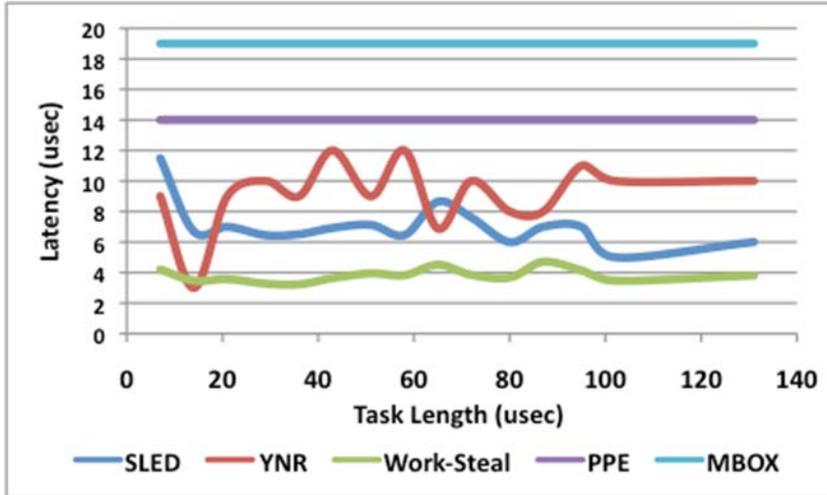
- Common execution model: off-loading
- Not everything can be processed on accelerators (example: inter-node communication)



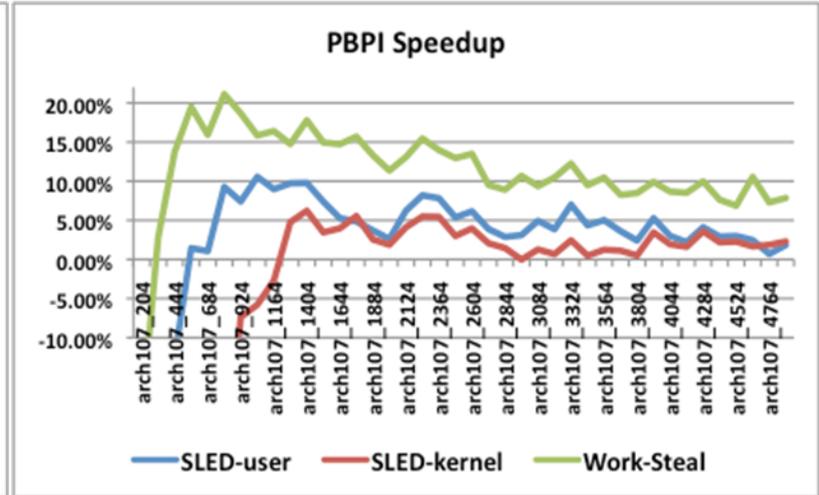
ProcSM - Accelerator Performance



Better



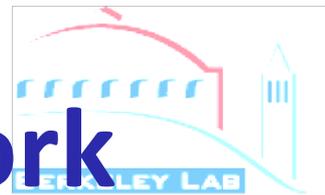
PPE-SPE Synchronization Overhead



Better

- ProcSM enables all-to-all mapping of processes to accelerators
- All-to-all mapping enables work stealing execution model
- Main Core <-> Accelerators synchronization latencies:
 - Work-Steal (UPC) 3us
 - SLED 7us
 - YNR 10us
 - MBOX 19us

Conclusions and Future Work



- We implemented Process/shared memory support for UPC runtime! (This work started before I joined the lab)
- ProcSM outperforms Pthreads in many cases!!
- ProcSM improves interoperability
- ProcSM currently supported within an SMP node, and in a distributed environment for Infiniband and MPI networks
- Future Work
 - Merge ProcSM with new collectives
 - Pinpoint the reasons for performance increase with ProcSM
 - Investigate UPC_all_alloc() behavior
 - Investigate hybrid execution: Processes + Pthreads, MPI+UPC ...
 - Confirm ProcSM performance on various architectures using additional benchmarks
 - Extend Language/Runtime support for accelerators



Thank You!