Data-Movement and Control Substrate for Scientific Applications

Induprakas Kodukula

Nikos Chrisochoides

Keshav Pingali

This research was supported by a grant from Cornell Theory Center, which receives major funding from NSF and IBM, New York State and members of its Corporate Research Institute

Organization

Why is this interesting?

System Architecture and Overview

Performance numbers

Related work

Conclusion

Interest - API

- API for Threads + One-sided communication
- Minimalist interface
- Homogeneous and heterogeneous platforms
- Proposed by PORTS consortium

Significant pieces of API

ports_thread_create

ports_thread_yield

ports_thread_exit

ports_newack

ports_testack

ports_waitack

ports_mycontext

ports_numcontexts

ports_get

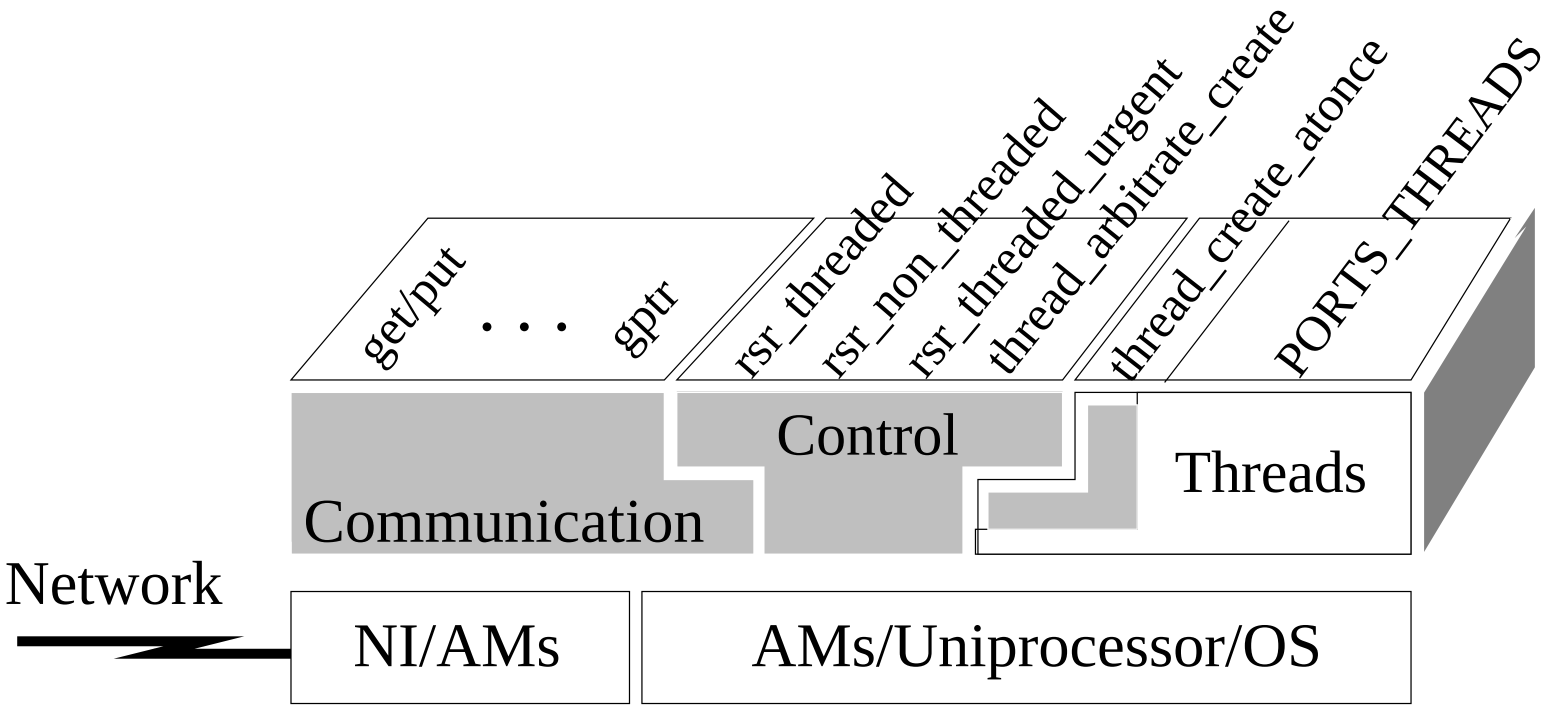
ports_put

ports_rsr

Interest - Implementation

- Homogeneous platform (SP-2)
- Assumptions about homogeneity minimized
- Lean and efficient
- Proposed PORTS functionality implemented
- Additional functionality for scientific code

System Architecture



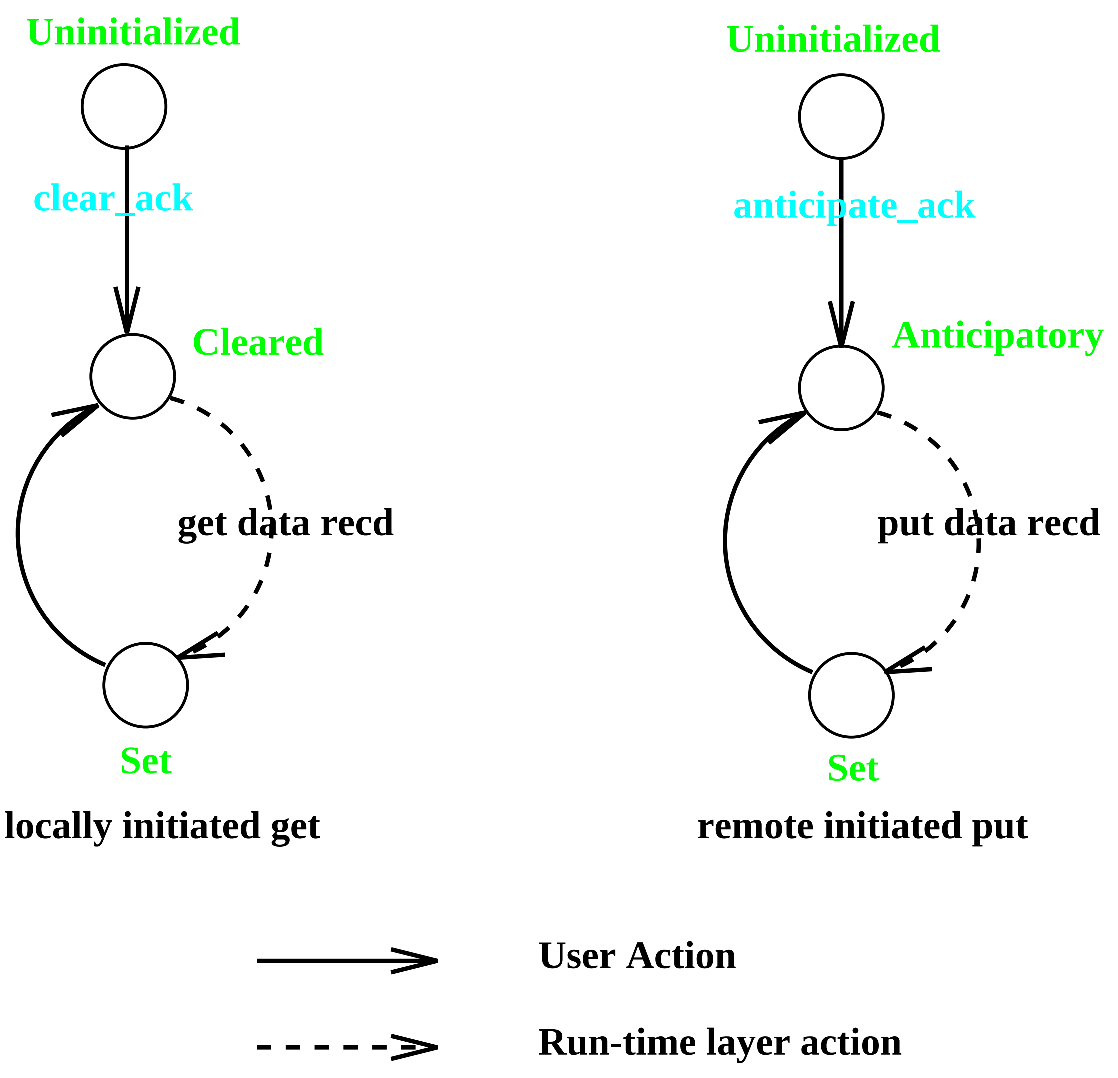
Threads package

- User-level core QuickThreads (SP-2 port Chichao, et al.)
- Non preemptive core with no policies
- Scheduling policy provided by threads package
  - *dmcs_thread_create*
  - *dmcs_thread_yield*
  - *dmcs_thread_exit*
  - *dmcs_thread_create_atonce*

Communications Module

- On top of Active Message implementation on the SP-2 (Chichao, et al.)
- Global pointers
  - $dmcs\_mycontext, dmcs\_numcontexts$
  - $dmcs\_localpointer$
- Acknowledgement (ack) variables - to manage asynchrony
  - $dmcs\_newack$
  - $dmcs\_testack, dmcs\_waitack$
- Data transfer - asynchronous
  - $dmcs\_get(lptr, gptr, local\_ack)$
  - $dmcs\_put(lptr, gptr, remote\_ack)$

State changes for acknowledgement variables



Control Module

- Remote Service requests
  - Action - $function\_pointer$ or $index$
  - Destination - $context$ or $global\_pointer$
  - User buffer argument
  - Non threaded or threaded
    - Low priority thread
    - High priority thread
- Thread-count based load balancing
  - Destination chosen at creation
  - Running thread never migrated

Why is this interesting?

System Architecture and Overview

- Performance numbers

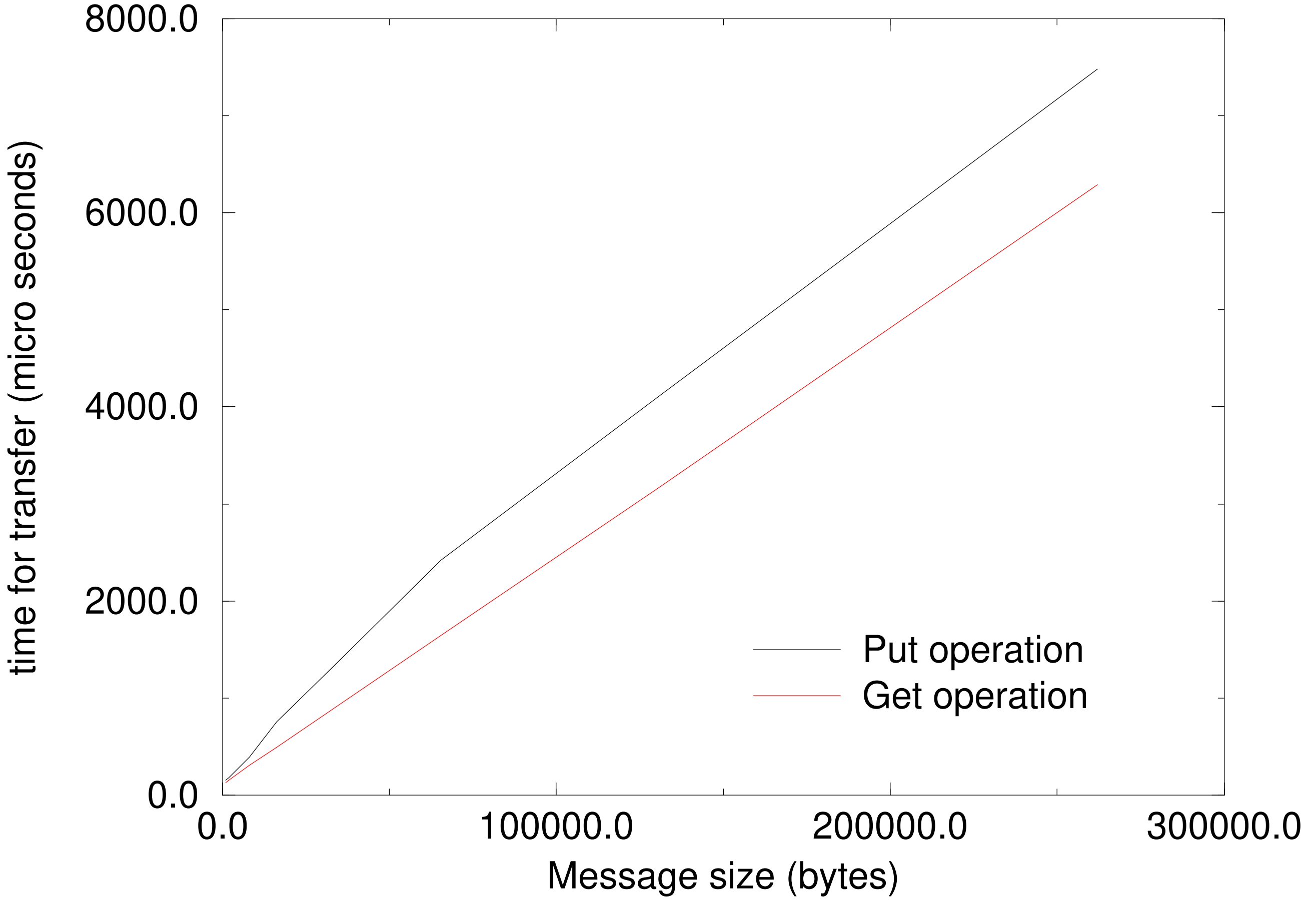
Related work

Conclusion

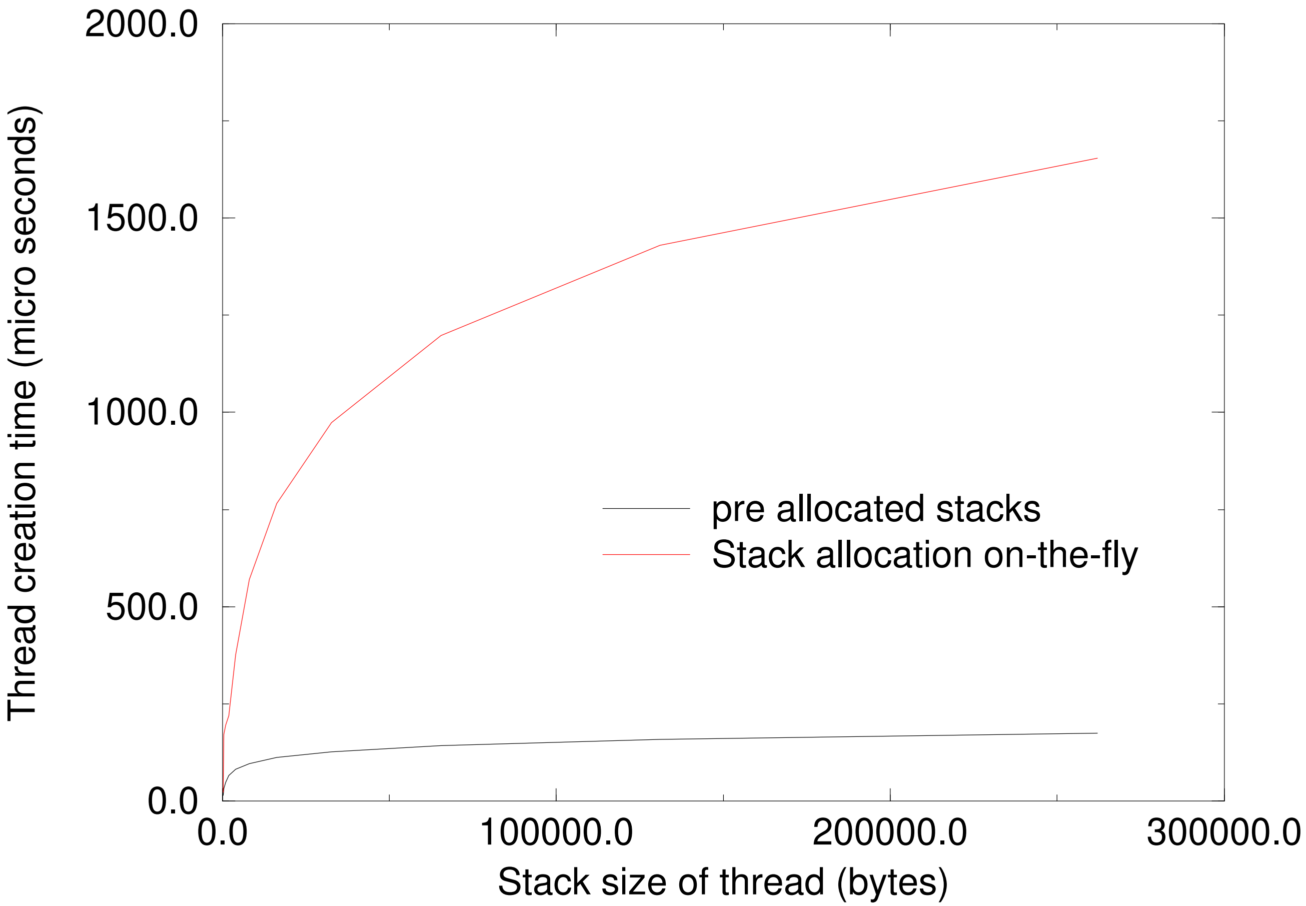
Basic system parameters

- Thread creation time = $12 \mu s$
- Context switch time = $5.5 \mu s$
- Peak Data transfer bandwidth = $33.6 MBytes/sec$
  - Peak underlying bandwidth = $35 MBytes/sec$
- One-way latency for a 0-byte message $29 \mu s$.
  - 10% overhead over active message latency of $26 \mu s$.
- Non-threaded null remote service request = $31 \mu s$
  - Nexus latency = $79 \mu s$, 40% over MPL

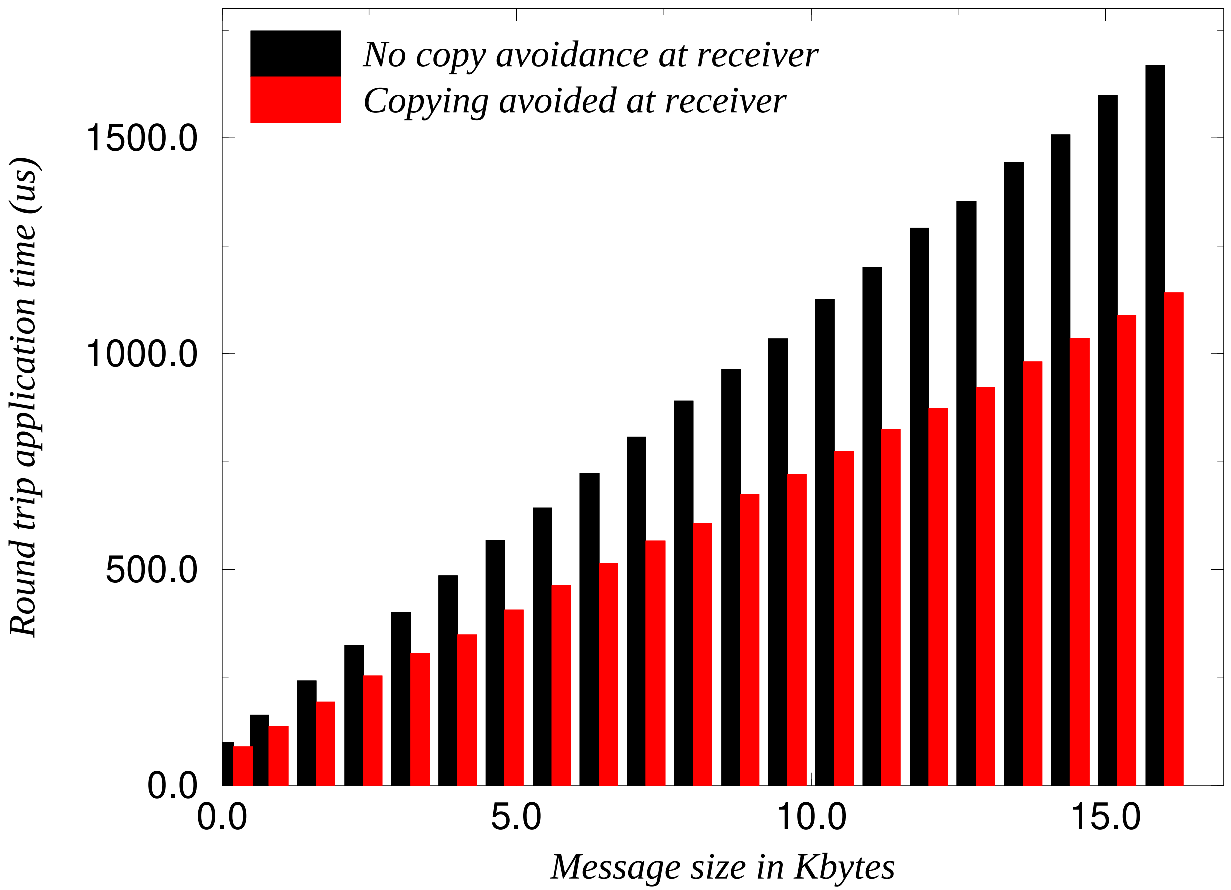
Communication overhead vs message length



Thread creation performance



Performance gains by exploiting homogeneity



Ongoing work

- User defined RSR bindings
  - RSR index bound to different functions at sender and receiver
  - Useful in adaptive mesh refinement
- Support for AMR
  - Support for general data distributions
  - Support for general communication patterns
  - Exploring more general load-balancing

Current users

- Task-parallel implementation of Bowyer-Watson mesh generation algorithm
- Runtime support system - adaptive mesh refinement
- Implicit load-balancing - mobile pointers
- SGI port used in scientific visualization application

Why is this interesting?

System Architecture and Overview

Performance numbers

- Related work

Conclusion

Our contributions

- Integration of threads and communication in a separate module
- 0/1 Priority mechanism at thread creation time
- Minimize assumptions regarding homegeneity
- Prioritization of remote service requests
- Exploit homogeneity
- Anticipate mechanism for a put operation

- Other run-time substrates
  - Nexus - RSR based, primarily for heterogeneous systems
  - Chant - Integrated thread-to-thread communication
  - Tulip - Hybrid mechanism

Conclusions

- Implementation of API for threads + one-sided communication
- Very lean and efficient
- Separate modules for control and communication
- Plug and play with different modules
- Exploit homogeneity for better performance