Threads, I/O, and Distributed Computing with PVM

Vaidy Sunderam
Emory University
`vss@mathcs.emory.edu`

Overview

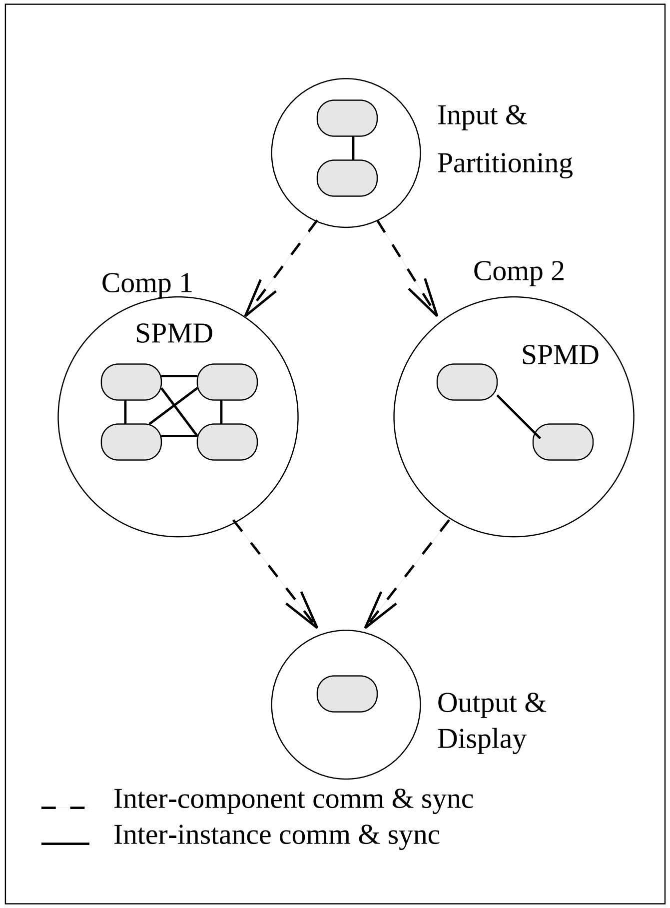
- Scope: Evolution, ongoing research, and futures - in the context of the PVM system

- Outline
  - Overview and Background
  - The Threads Subsystem
  - Parallel I/O for PVM
  - Concluding Remarks

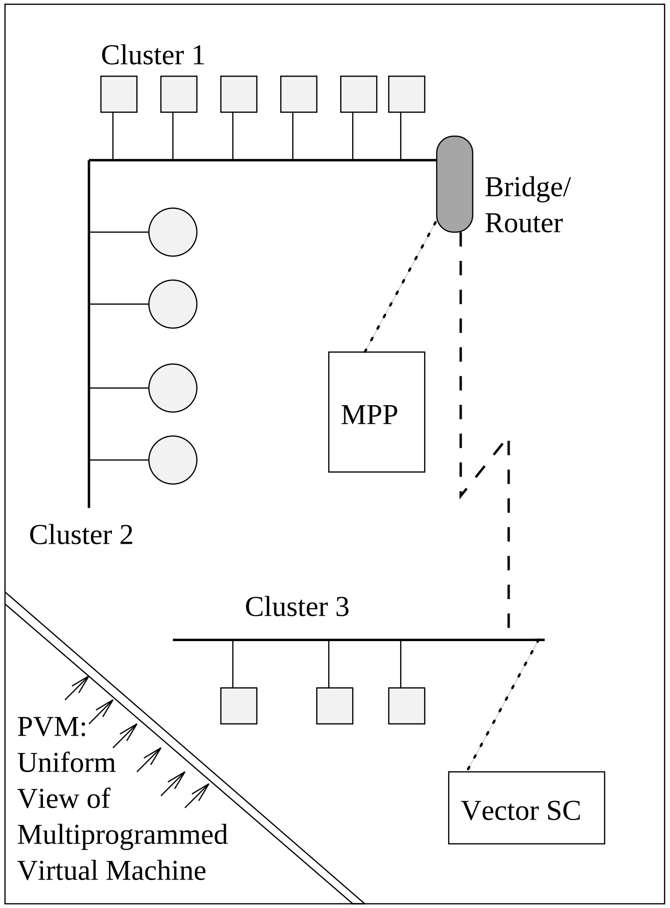
Background

- The PVM System
  - 5 years old and going strong!
    - First version - experimental prototype, summer 1989.
    - Version 2 (first public release) - March 1991; V3 released in February 1993.
    - Now - ongoing experimental systems research project at ORNL, UT, Emory, CMU.
    - Software in widespread use; basis for external research projects.
  - Basic themes:
    - User configured host pool --> virtual machine concept
    - Translucent access to hardware
    - Process based computational model
    - Explicit message passing paradigm
    - Heterogeneity support
    - Multiprocessor support

• PVM Computing Model



(a) PVM Computation Model



(b) PVM Architectural Overview

This page used for formatting purposes only

- Applications
  - Predominantly scientific computing applications, numerical algorithms.
  - Significant use for teaching/learning; isolated non-scientific and commercial use.
  - Facilities needed for "mainstream" computing with PVM
    - General purpose distributed computing facilities
    - "Service" oriented computing, transparency
    - Data storage and handling, distributed/concurrent I/O
    - Transaction facilities, concurrency control, locking
    - Security, authentication, access control, accounting
    - Load balancing, failure resilience, admin tools
  - Ongoing research and development projects at various institutions are addressing these (and other) issues.
  - TPVM: Threads based concurrent computing with PVM
  - PIOUS: A parallel and distributed I/O system

An Alternative Paradigm

- Threads-based Network Concurrent Computing
  - Multi-threaded architecture for concurrent heterogeneous computing; LWP's are units of parallelism/scheduling
  - Prospects: adaptive parallelism, increased asynchrony, alternate computing model
  - Rationale and Motivations
    + Trends in WS, servers : small-scale SMMP's with 1-8 processors
    + Potential for smaller granularity; more natural problem decomposition
    + Greatly increased potential for latency hiding, overlap of communication and communication
    + Enables straightforward programming of asynchronous concurrent computation
    + Facilitates "exportable services" and self-scheduled processing entities

- The TPVM system
  - An experimental threads subsystem/architecture layered on PVM
  - Address performance issues, and explore "service" based concurrent computing methodology
  - Design Goals
    + Overcome rigid binding: In most DME’s, process to processor and workload to process binding is strong, early, and inflexible.
    + Support "process-oriented" as well as "service-oriented" programming models for wider applicability, usability
    + Enable translucent/explicit global memory emulation
    + Adaptive parallelism: respond to changing environment or changing external influences and loads

- Process-Oriented Model
  + Application sub-algorithm is a collection of cooperating threads
  + Facilities for thread spawning & management; each thread has unique id
  + Communication interface : uniform message passing — retain explicit message passing paradigm
- Exportable Services Model
  + "Shells" distributed through VM contain latent computational services
  + Dynamically instantiated as lightweight processes -- when needed, at appropriate location
  + Shades of RPC, dataflow, active messages, . . .
- Remote Memory Access via Exportable Handles
  + Programming/system framework for network concurrent computing with asynchronous access to remote memory
  + Basis for virtual shared address space or remote memory copy
  + Strongly typed; operation in heterogeneous networked environments

- Programming Interface
  - Explicit DM message passing model
    - + Processes initiated via traditional PVM mechanisms
    - + **`tpvm_export("matmul", mmproc,..);`** establishes set of exported services
    - + Multiple instances (with load-balanced scheduling) activated via **`tpvm_spawn("matmul",..,<ttids>);`**
    - + Threads cooperate via explicitly addressed message passing i.e. **`tpvm_send(ttid, . . .)`**, **`tpvm_recv(. . .)`**, etc
  - Performance

Problem (size)	CPUs/Processes		CPUs/Threads		
	4/4	16/16	4/16	16/64	16/36
Matmul (500x500)	201	78	182	94	60
Matmul (1000x1000)	1618	756	1610	710	663
Sorting (2M ints)	1423	366	1201	351	309

– Trigger-messages and event-driven computation

+ Firing rules based on message-tag conjunctions, e.g.

```
tpvm_export("matmul", mmproc,
<flags>, "88" && "99");
```

+ `tpvm_invoke("matmul", 88, <flags>);` partially satisfies firing rule.

+ Thread initiated at "best" location when all rules satisfied; messages delivered

– PMR Example: square grid of processes; each responsible for square subblock of C; each process *P(i,j)* :

```
tpvm_export("mm",..,tA(i,*)&&tB(*,j))
pvm_initsend(); pvm_pkdouble(A(i,j));
for i = 1 to dim
     tpvm_invoke("mm", tA(i,j), ...);
```

– Performance

Problem	CPUs/Processes		CPUs/Threads			CPUs/Threads	
(size)	4/4	16/16	4/16	16/64	16/36	4/?	16/?
Matmul (500x500)	201	78	182	94	60	194	62
Matmul (1000x1000)	1618	756	1610	710	663	1582	640
Sorting (2M ints)	1423	366	1201	351	309	1167	309

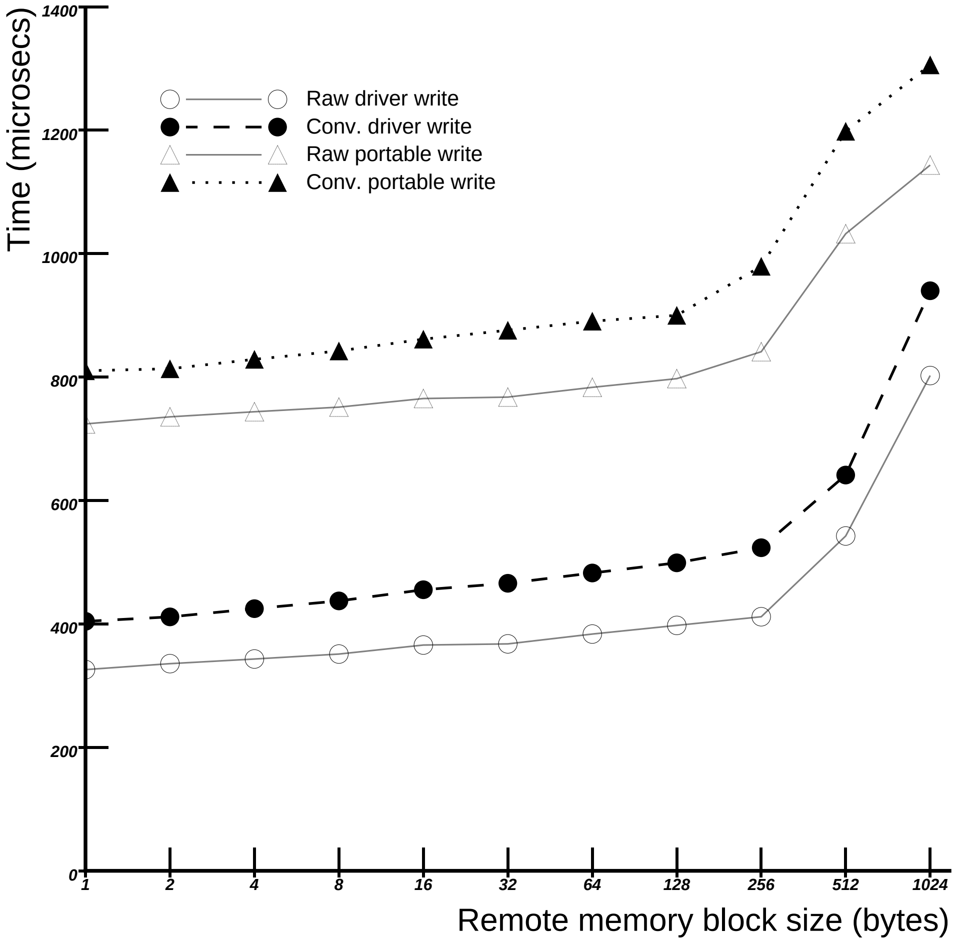
– Remote Memory Services

+ Data areas in PVM processes exported using
`tpvm_exportd("vecA", myid, dataZ, COMPLEX8, 100, RW);`

+ Numeric "version" number (or section, subblock number) for partitioning global virtual data structure

+ Remote processes/threads access exported data using
`tpvm_read("vecA", partno, tempZ, COMPLEX8, 100);`
`tpvm_write("vecA", partno, tempZ, COMPLEX8, 100);`

– Preliminary Performance Results



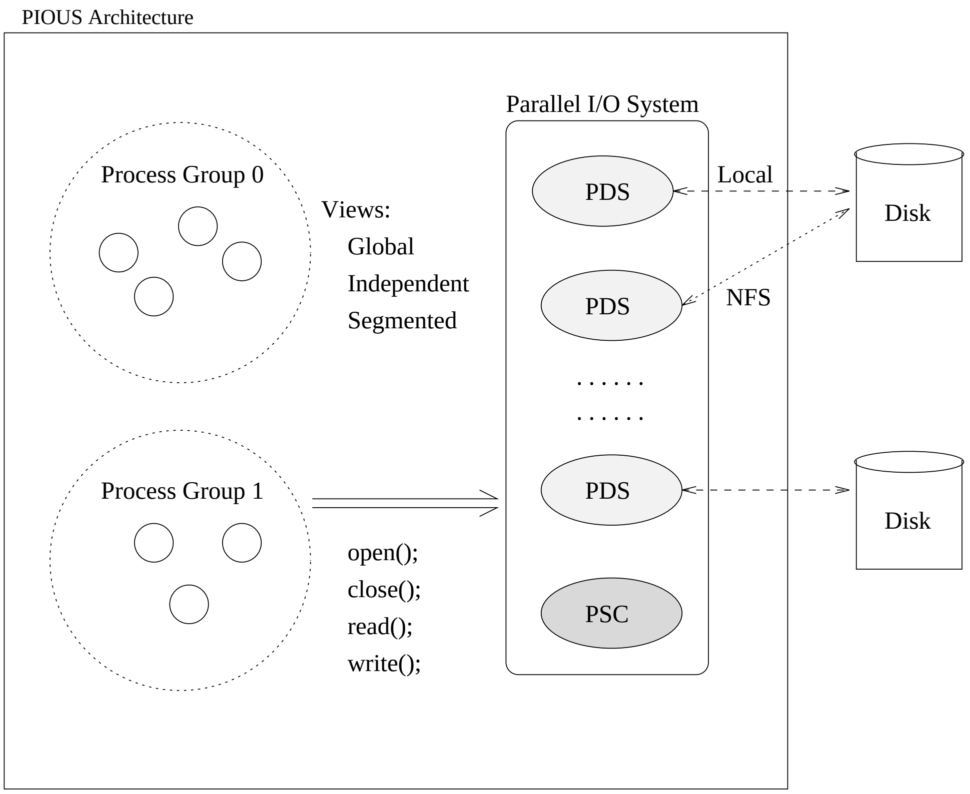
TPVM Remote Memory Access Times (usec)

This page used for formatting purposes only

Parallel I/O in PVM

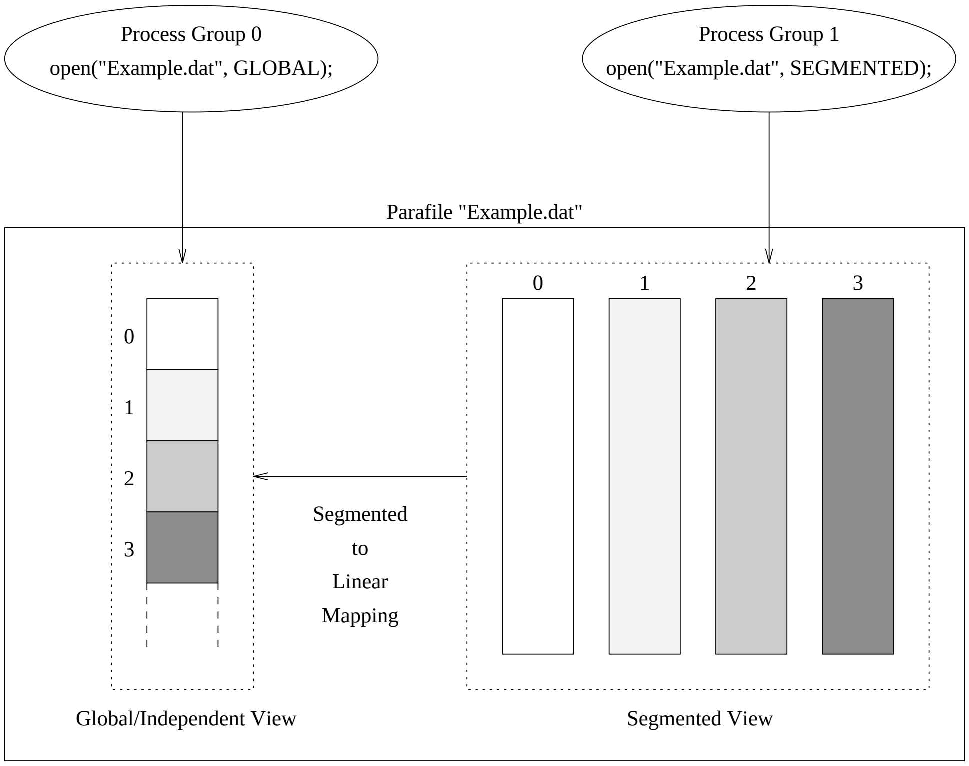
- The PIOUS Architecture and System (with Steve Moyer)
  - Motivations
    - + DCE's generally lack a unified parallel I/O system
    - + Potential for application domains greatly enhanced
  - Goals
    - + High-performance, scalability, reliability
    - + File model and interface for parallel computation
  - Architecture Principles
    - + Transport and native file system independence
    - + Asynchronous model of operation
    - + Data declustering
    - + Integrated concurrency control and fault tolerance mechanisms
    - + Access mechanism and policy independence

- PIOUS Software Architecture



- Software components
  + One PDS/(disk) machine (may be NFS)
  + Transaction-based access to local files
  + Local and independent operation
  + Global PSC - only for major events
- Client libraries
  + Translate operations to PDS/PSC requests
  + Implements file system model and interface

- PIOUS File System Model & User Interface
  - Parafile objects; interface supports coordinated interaction
  - Groups, Views, and Segments



- Interface and Semantics
  - PIOUS file views
    - Global: linear sequence, shared pointer
    - Independent: linear sequence, local pointer
    - Segmented: declustered, local ptr segment access
  - Consistency semantics
    - Sequential consistency within/between process groups
    - Selective file locking at group level
  - Selectable fault tolerance
    - Volatile: concurrency control only
    - Stable: concurrency control and system failure tolerance

- PIOUS User Interface

  – Supports the defined model semantics; I/O functions and maintenance functions.

  – File Primitives

```
pious_pfopen(path, view, faultmode, . . .)
pious_pfopens(servers, path, view, faultmode,

pious_open(path, oflag, mode)

pious_close(fd)

pious_fileconf(fd, name)
```

  – I/O Primitives

```
pious_read(fd, buf, nbyte)
pious_reado(fd, buf, nbyte, offset)

pious_write(fd, buf, nbyte)
pious_writeo(fd, buf, nbyte, offset)

pious_lseek(fd, offset, whence)
```

- Maintenance Primitives

```
pious_chmod(path, mode)
pious_chmods(servers, path, mode)

pious_unlink(path, mode)
pious_unlinks(servers, path, mode)

pious_mkdir(path, mode)
pious_mkdirs(servers, path, mode)

pious_rmdir(path, mode)
pious_rmdirs(servers, path, mode)
```

- Transaction Primtives

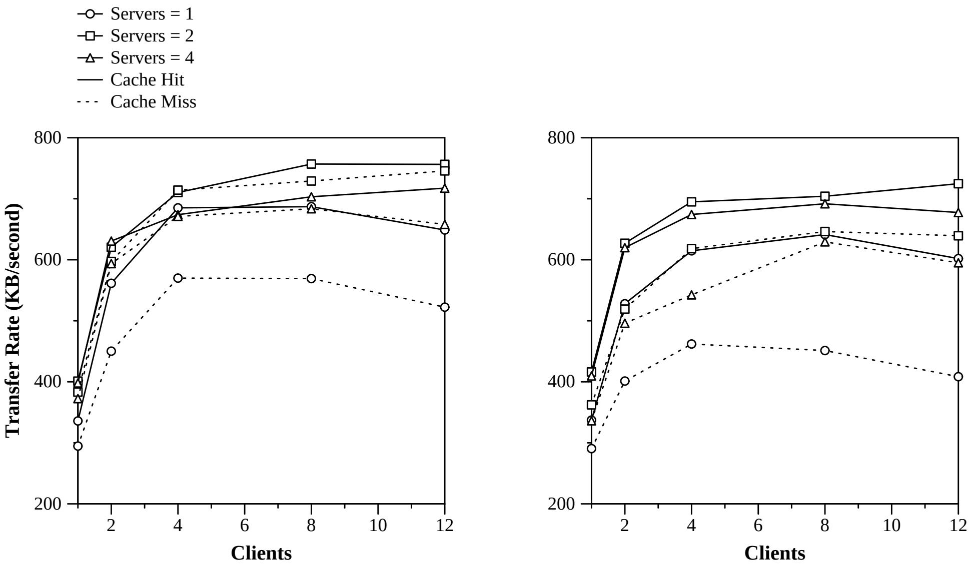
```
pious_tbegin(faultmode)

pious_tabort()

pious_tend()
```

- Record support
  - + Interface insufficient for heterogeneous environment: operates on uninterpreted bytes, application must perform data conversion
  - + Possible solutions: record descriptor (e.g. MPI), typed/self-describing files

- Prototype Performance Study

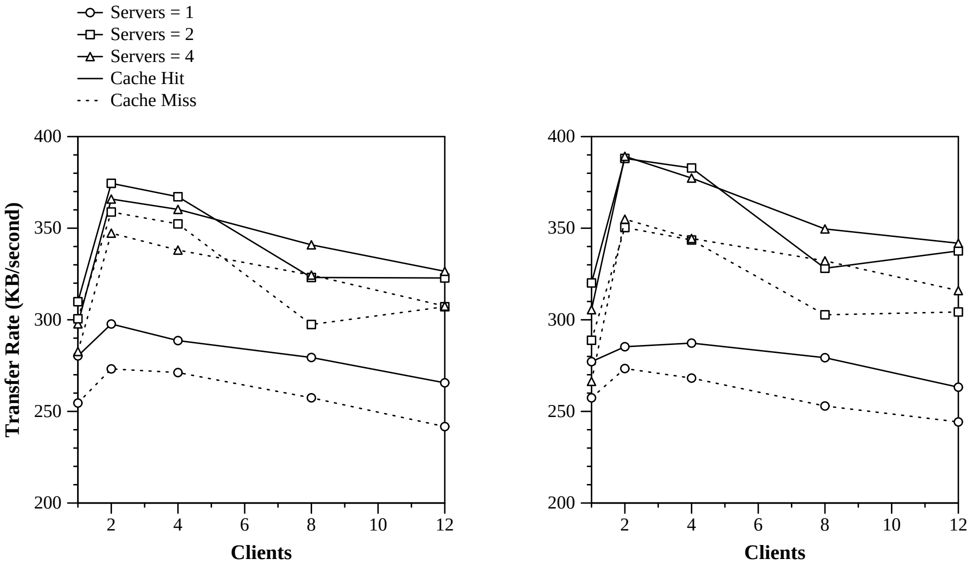
  – PVM on Ethernet; clients - SS2 (SunOS5.3); servers - SS2 (SunOS5.3), 16MB, Maxtor 7345S.

  – Partitioned/Volatile



(a) Read (b) Write

  – Self-scheduled/Volatile



(a) Read (b) Write

Concluding Remarks

- Heterogeneous Concurrent Computing
  - Viable technology; Moderate to very good performance and functionality
  - Now in the mainstream; potentially standard
- The PVM Software System
  - Evolved from proof-of-concept to production use; robust, minimal, kernel substrate; add-ons for functionality & performance
  - Widespread adoption, impact on HP computing
- Ongoing and Future work
  - Infrastructure for expanding applicability - new OS'es, I/O, fault-tolerance, transactions, objects, threads.
  - Optimize performance - multithreading, fiber/gigabit network support, dynamic scheduling & load balancing
  - Tools - for management & administration, development, debugging, profiling, visualization
  - Towards a "timeshared network computing system ..."