PVM : Past, Present, and Future

Vaidy Sunderam

Emory University
Atlanta, GA 30322, USA
+1 (404) 727-5926
vss@mathcs.emory.edu

Research supported by NASA Ames - NAS RNR under NAG2-282

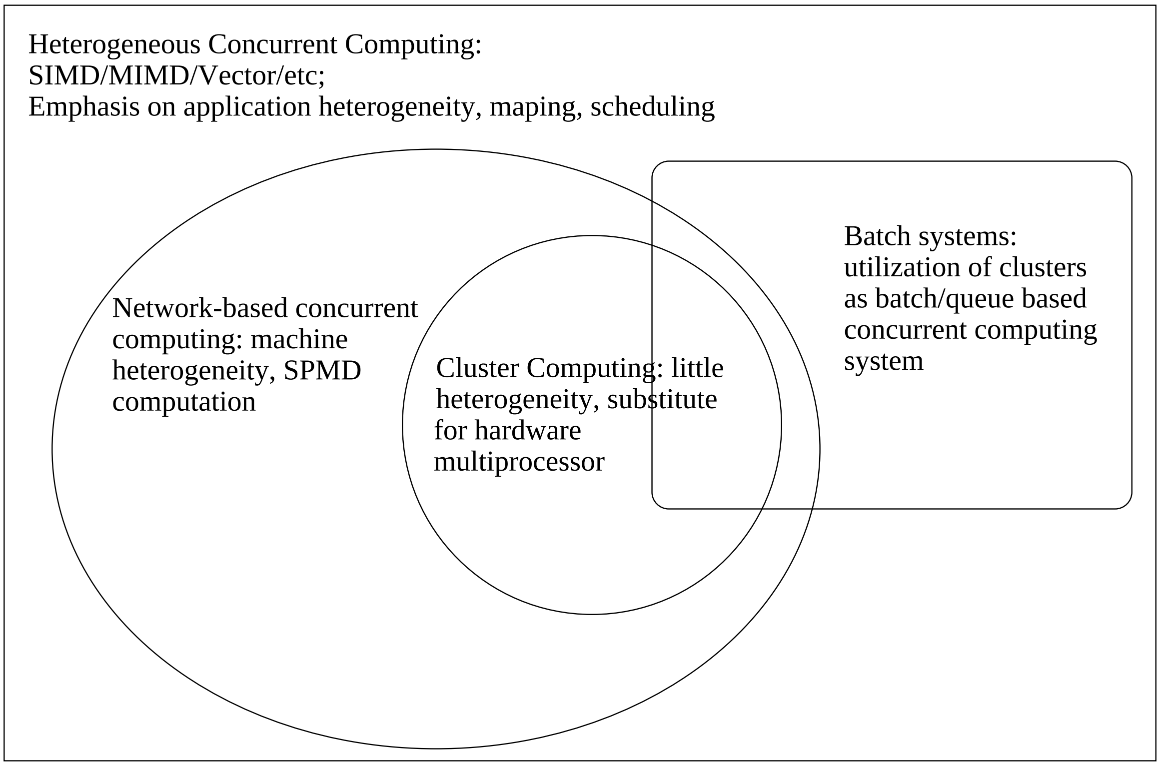
Overview

- PVM for Heterogeneous Concurrent Computing
  - Effective use of multifaceted computing environments
  - Applications : sub-algorithms with varied characteristics and requirements
  - Functionality, cost-effectiveness, performance
- Scope: The PVM software system for concurrent computing on heterogeneous networked environments.
- Outline
  - Introduction and Overview
  - Evolution of the PVM System
  - Current Status and Recent Results
  - General Purpose Distributed Computing
  - Futures, Ongoing Work, Concluding Remarks

Background

- Traditional Parallel Applications:
  - Precise and well defined; regular computation and communication patterns.
  - Suitable for scientific supercomputing on symmetric, homogeneous hardware architectures.
- Evolving Applications :
  - Complex systems with many interacting components.
  - Varied characteristics (languages, models, etc).
  - Processing + I/O, transactions, fault tolerance.
  - Not well suited to closed, proprietary systems.
- Examples :
  - Global Environment Simulation: Fluid flow analysis (vector); Transport modeling (DMM or WS network); Temperature effects (scalar); User interface (graphics front end).
  - Integrated Trading System: Yield simulations (DMM); distributed TP (loosely coupled clusters); high-volume IO (data servers); visual display/interaction (graphics).

- Typical Computing Environments
  - High speed network, gatewayed to remote resources.
  - Workstations, scalar servers, large backing store, substantial support services
  - Multiprocessors, occasional supercomputers.
- Heterogeneous Concurrent Computing
  - General, flexible, concurrent computing environments built on networked collection of heterogeneous machines.
  - Technologies and their relationships:



This page used for formatting purposes only

- Heterogeneous Network Computing
  - Evolution and historical perspective:
    - + Early-mid 80's : Workstations (~1-3 MIPS, few $K) and LAN's (~1 MBs)
    - + Relative abundance of aggregate resources
    - + Remote execution, farm computing, migration etc
  - Currently, basic scenario is the same, but
    - + Workstations - 100's of MIPS/MFLOPS
    - + Multiprocessors, occasional supercomputers are accessible
    - + Large and varied resources, convenient interfaces
  - Now is the time . . .
    - + Imminent increase in network capacity
    - + Grand challenge applications - immense and varied resource requirements
    - + Increasing commercial usage of distributed and concurrent computing
    - + High potential (functionality & performance), little or no (initial) investment; relatively low development, administrative effort.

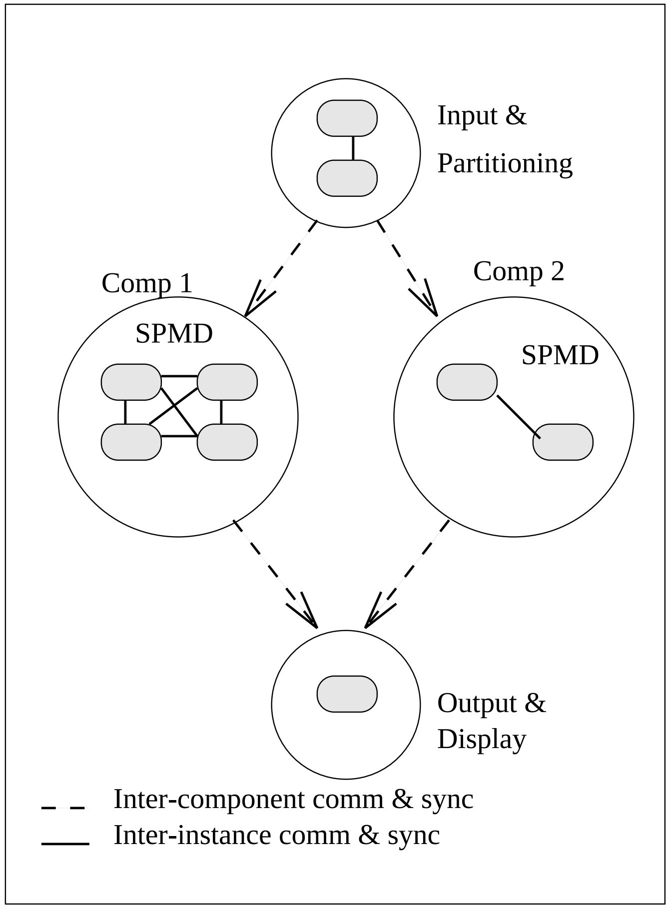
- Systems, Environments, and Languages
  - Distributed Operating Systems
    - Examples: Mach, Chorus, Amoeba
    - Close to hardware, low level implementation
    - Issues: File services, process facilities, security, management
  - Languages
    - Orca, Concurrent C, Jade, Occam, Linda
    - Some involve new paradigms, external support
    - Issues: Model, translation, applicability.
  - Environments
    - Express, PVM, Linda, ISIS
    - Flexible and general, good performance
    - Development, administrative, debugging, and profiling support
    - Well suited for high-performance scientific and general applications
    - Support heterogeneous requirements well
    - Issues: Support tools, efficiency, portability

- Applications
  - CFD using interface methods in 3-D, volume visualization (Suny SB, NASA Ames), CFD, aircraft design (NASA Lewis)
  - Finite element methods (McGill, Convex), 3D potential/elliptic solvers, Hypersonic flow simulations (NASA Larc, Langley)
  - Fast-response database query system (Boeing); structure diagram storage and retrieval (Chemical Abstracts Service); concurrent searches of the protein databases (UCSD)
  - Computer aided tomography of soft x-ray emission (LLNL), Electron Gamma Shower - EGS Monte Carlo code (National Research Council of Canada), PET image reconstruction (Washington)
  - Teaching parallel programming, basis for research projects (Stanford - Jade, UT Austin - Code, Parallel Computer Algebra - OSU)
  - Distributed modeling (BT), Economic forecasting and yield analysis (Wall Street Firms).

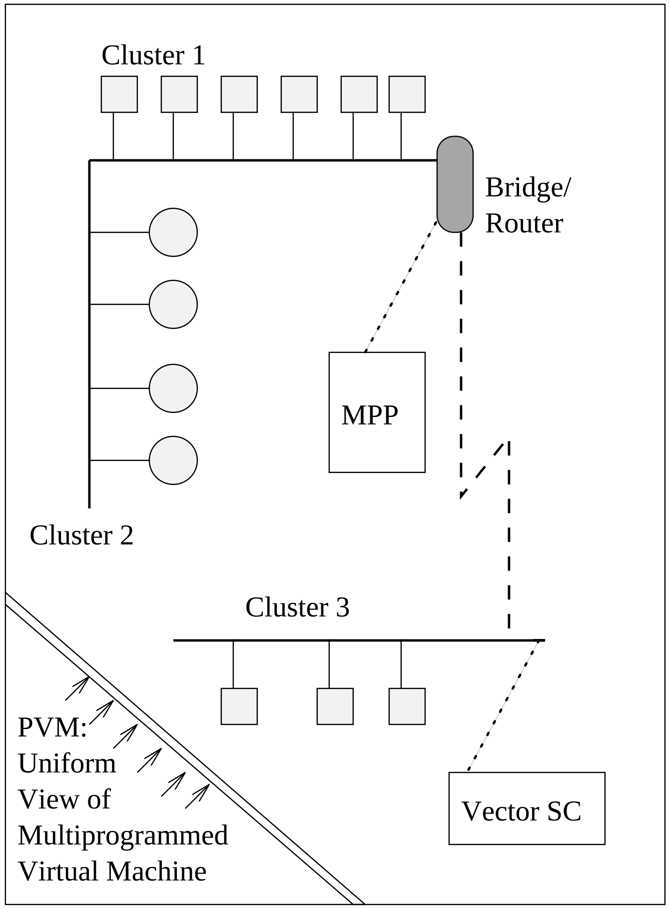
PVM System Overview

- Overall Theme
  - General, flexible, concurrent computing environment on networks of heterogeneous machines.
  - User interface primitives embedded in procedural languages for process management, communication, synchronization, and other functions.
  - Support software executes on participating network hosts; presents virtual concurrent resource.
- Design Goals
  - Applications:
    - + Straightforward, well understood paradigms.
    - + Graphical development tools.
    - + Debugging and profiling facilities.
  - System:
    - + Support for multiple architectures (esp. multiprocessors) and networks.
    - + Efficient distributed algorithms, multi-party protocols, failure resilience.
    - + Straightforward installation, maintenance, operating and administrative interface.

- PVM Computing Model



(a) PVM Computation Model



(b) PVM Architectural Overview

This page used for formatting purposes only

- PVM Software System
  - Public domain, full source availability, user installed; primarily Unix-based
  - Components: (1) daemons on host pool emulate parallel machine; (2) interface library presents cooperation and coordination primitives.
  - Applications: explicit distributed memory parallel programs, compiled for each architecture.
- PVM daemons
  - Free running Unix processes; standard Internet protocols for inter *pvmd3* communication
  - Reliability, ordering, and other protocol enhancements within PVM software
  - Process management via normal Unix mechanisms; facilities for dynamic VM re-configuration
- User library
  - Standard concurrent computing primitives; interaction via local *pvmd3* or direct
  - Algorithms for data fragmentation and reassembly, memory management; little state maintained.

Application Development

- PVM Programming Model
  - Application consists of *components*; executing instances are *processes* or *tasks*.
  - Process based model: unit of concurrency, identified by logical **`taskid`** (or symbolic name and instance number), no process-processor relationship
  - Collection of *component instances*, cooperating via explicit message passing and synchronization primitives
  - Asynchronous cooperating sequential processes: distributed threads, primarily message passing model
  - Dependencies and relationships implemented using control flow constructs in host languages . . .
  - Augmented by event/message based synchronization
  - PVM primitives for:
    + Process management (initiation, termination)
    + Communication (send, receive, broadcast)
    + Synchronization (barriers, rendezvous)
    + Miscellaneous (status checking, housekeeping)

- Partitioning and scheduling is explicit and manual
- Process Structure and Interaction
  - Processes must register to participate:
    `mytid = pvm_mytid();`
  - An enrolled process may spawn others:
    `pvm_spawn("factor", . . . );`
  - Synchronization

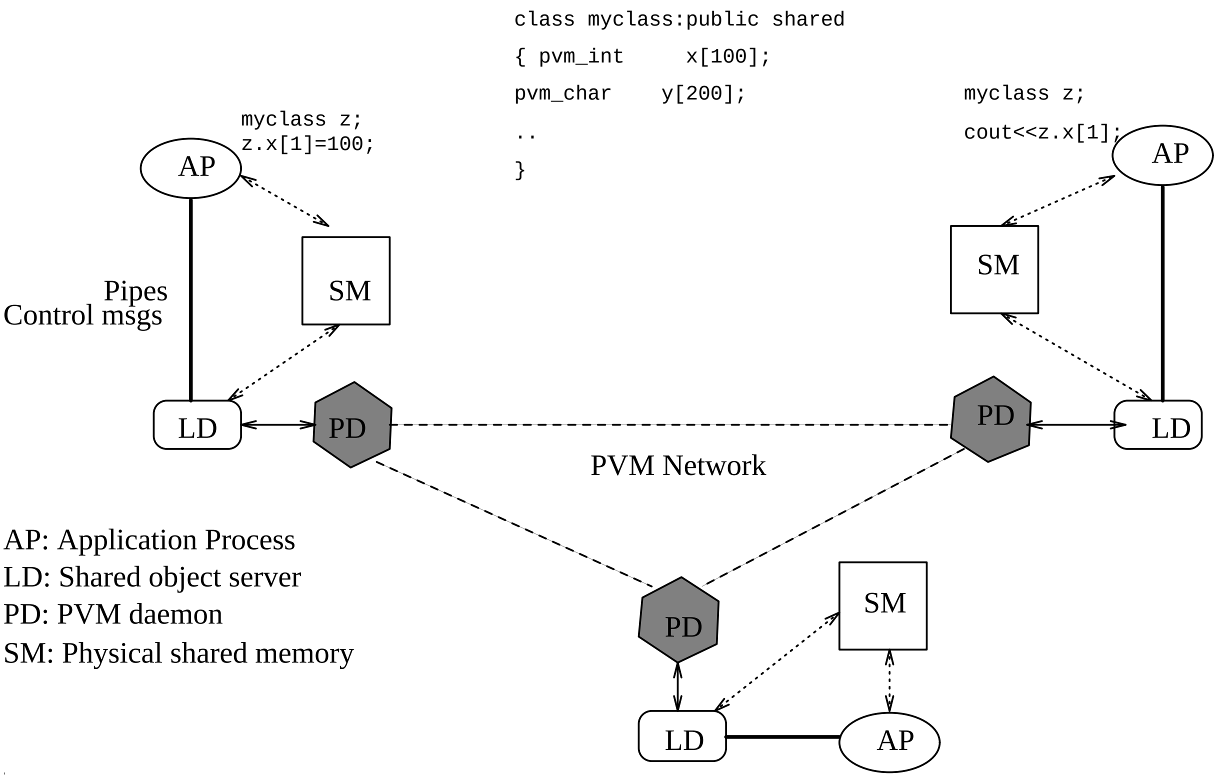
    ```
    pvm_barrier("ourgroup",8)
    pvm_recv(-1,EV)/pvm_mcast(wtids,EV)
    pvm_sendsig(wakeup_tid, SIGUSR1)
    pvm_notify(TaskExit,DTYP,1,tid)
    ```
  - Data transfer:

Sender	Receiver
`pvm_initsend(0);`	`bid = pvm_recv(-1,99);`
`pvm_pkstr("hello");`	`pvm_upkstr(chptr);`
`pvm_pkint(&n,1,1);`	`pvm_upkint(&k,1,1);`
`pvm_pkfloat(&pi,1,1);`	`pvm_upkfloat(&f,1,1);`
`pvm_send(tid[4],99);`	`rcvinfo(bid,&len,&type,&stid);`

– Process flow graph constructed by embedding appropriate variants in sequential, conditional, iterative control flow statements.

```
while (needmoresuns())
     pvm_spawn("matrix_factor",
     . . ,"SUN4", . .);
     . . .
     . . .
if (pvm_pstat(input_tid) == PvmNoTask)
     pvm_spawn("compute",
     . . , "CRAY", . .);
     . . .
     . . .
pvm_recv(-1, DATA_READY);
pvm_spawn("plot_data", . . ,
     "silicon.graphics.com", . .);
```

- Shared Object Programming
  - Virtual Shared Object Space, shared object programming using the C++ language
  - Transparent access to shared objects, consistency
  - Shared object structures to facilitate partitioning, work-load distribution
  - Performance comparable to native message passing implementations
  - Architectural Overview

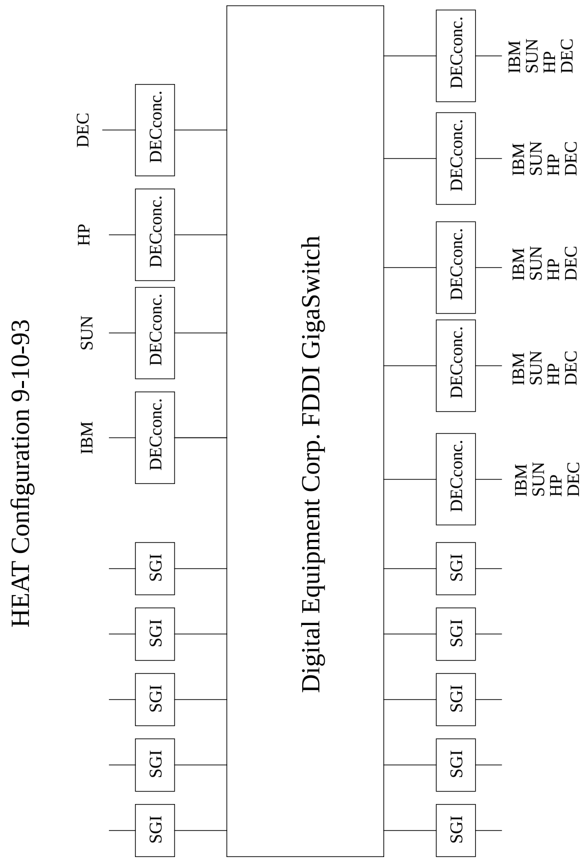


This page used for formatting purposes only

Performance Issues

- Inherent capabilities and limitations
  - Large grained applications : good performance, speedup
  - When communication latency is critical : poor performance, primarily communication software overheads
  - High communication volume : possible to achieve near-theoretical performance
- The NAS Parallel Benchmarks
  - Devised by NAS for benchmarking
    - + Five "kernels" & three simulated applications
    - + Rigorously test computation and data movement
    - + Representative of computational aeroscience applications
  - Porting the NPB kernels to PVM
    - + Derived from Intel versions and serial code
    - + First-cut implementation, no algorithmic optimizations
    - + Joint with Scott White & Anders Alund

- Performance measurements of NPB kernels on PVM
  - Tested Environments
    + Env #1: 16 SS1+ on Ethernet
    + Env #2: 8 RS6000 on FDDI
    + Env #3: 8 SGI on Gigaswitch



This page used for formatting purposes only

– The Five Kernels

Bench-mark	Prob. size	Memory (MB)	YMP/1 (secs)	Operation count	
Embar. parallel	2^{28}	8	126	2.67×10^{10}	211
V-cycle MG	256^3	453.6	22	3.90×10^{09}	176
Conj. Grad.	$2*10^6$	83.2	12	1.51×10^{09}	127
3-D FFT PDE	256^2*128	343.2	29	5.6×10^{09}	196
Int. sort	2^{23}	248.8	11	7.81×10^{08}	N/A

– V-cycle MG on PVM

Platform	Time (secs)	Comm. vol.	Comm. (secs)	No. of msgs
16 SS1+ (Eth.)	198^*	96MB	154	2704
8 RS6K (FDDI)	229	192MB	162	1808
8 SGI (GSW)	264	192MB	112	1808
Cray Y-MP/1	22	N/A	N/A	N/A
i860/128	8.6	?	?	?

– Conjugate Gradient on PVM

Platform	Time (secs)	Comm. vol.	Comm. (secs)	No. of msgs
16 SS1+ Enet	701	370 MB	480	37920
4 RS6000 FDDI	285	130 MB	192	7116
9 SGI Gswitch	130	250 MB	101	19756
Cray Y-MP/1	12	N/A	N/A	N/A
i860/128	8.6	?	?	?

– Integer Sort on PVM

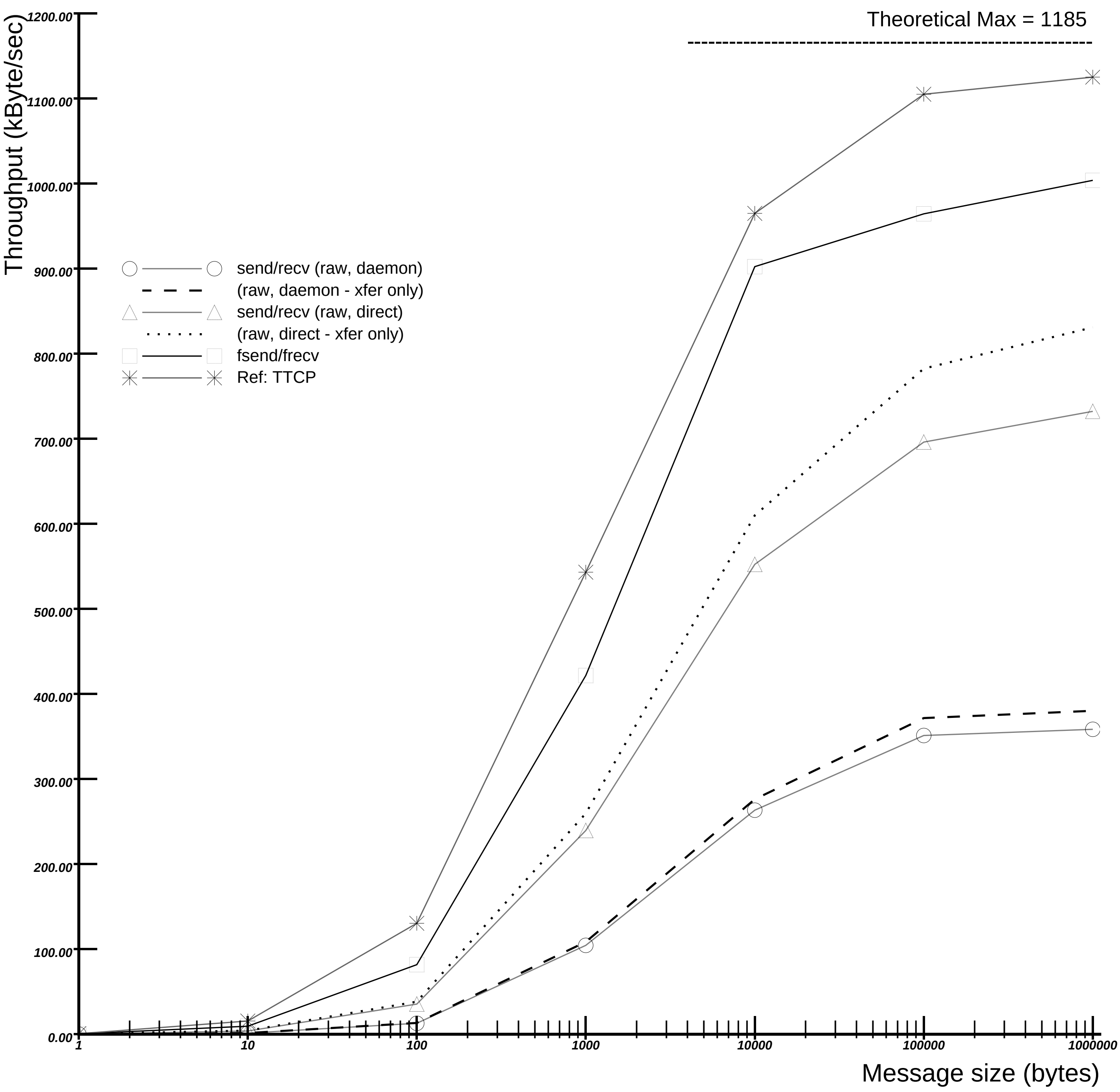
Platform	Time (secs)	Comm. vol.	Comm. (secs)	No. of msgs
16 SS1+ Enet	607*	150 MB	595	10115
8 RS6000 FDDI	674	560 MB	610	2491
8 SGI Gswitch	770	560 MB	720	2491
Cray Y-MP/1	11	N/A	N/A	N/A
i860/32	26	?	?	?
i860/64	17	?	?	?
i860/128	14	?	?	?

– 3-D Fast Fourier Transform on PVM

Platform	Time (secs)	Comm. vol.	Comm. (secs)	No. of msgs
16 SS1+ Enet	717*	420 MB	502	3450
8 RS6000 FDDI	645	1500 MB	395	826
8 SGI Gswitch	1070	1500 MB	516	826
Cray Y-MP/1	29	N/A	N/A	N/A
i860/64	21	?	?	?
i860/128	10	?	?	?

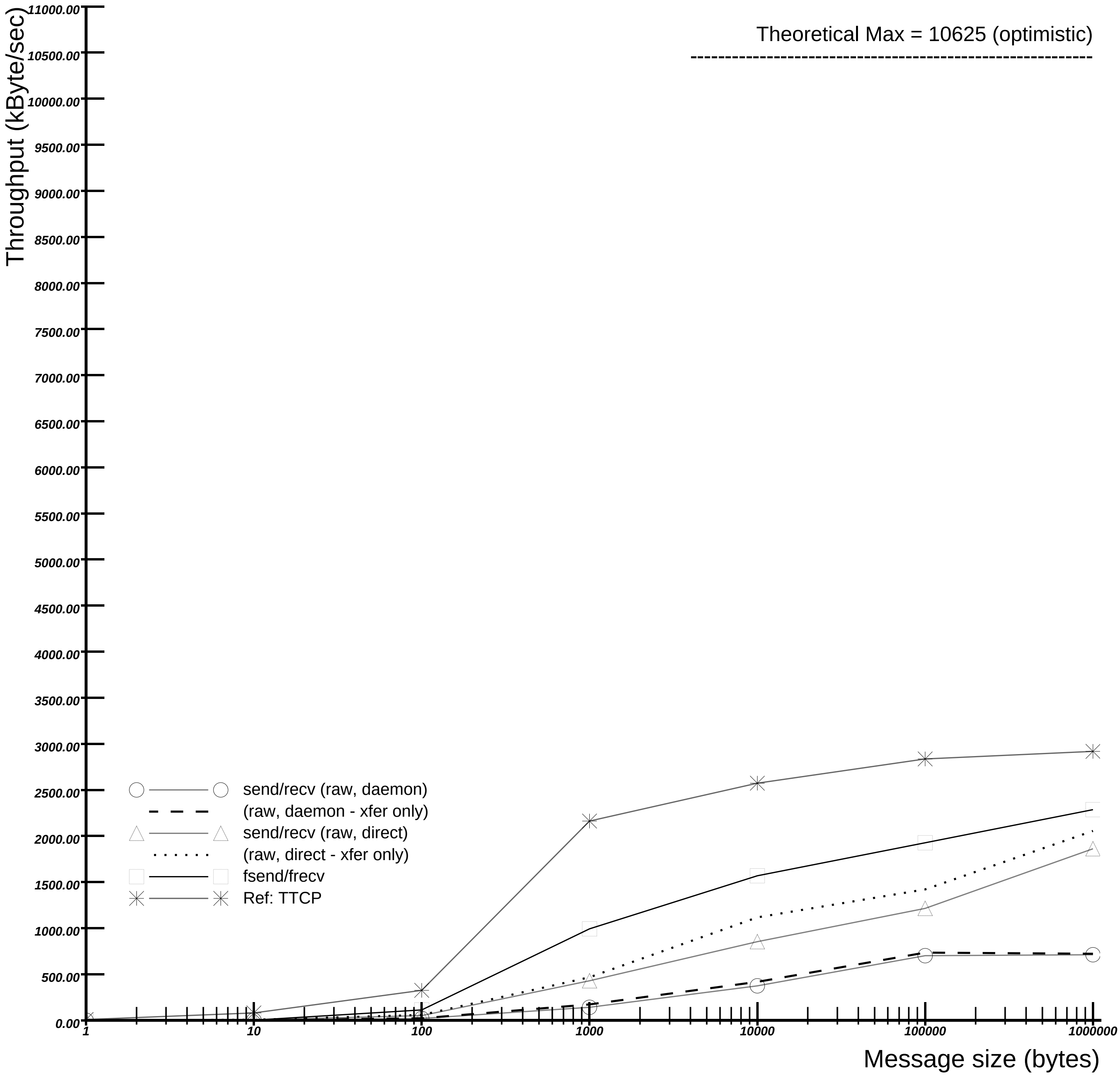
- Analysis of Performance
  - Performance levels generally poor to mediocre
  - Large volume communication attains medium throughput
  - High frequency communication results in poor utilization
  - Other observations: Collisions, load imbalance
- Enhanced Communication Scheme
  - Design Aspects
    - High BW for large msgs; low latency for small msgs
    - Avoid 3-stage communication
    - Eliminate copying/packing
    - Deadlock avoidance, asynchrony
  - Implementation
    - User buffers sent directly; memory allocation on receive
    - Optimized OS parameters for each machine/network
    - Connection management, multiplexing, deadlock avoidance schemes.

- Raw, point-to-point performance
  - SS1+ on Ethernet



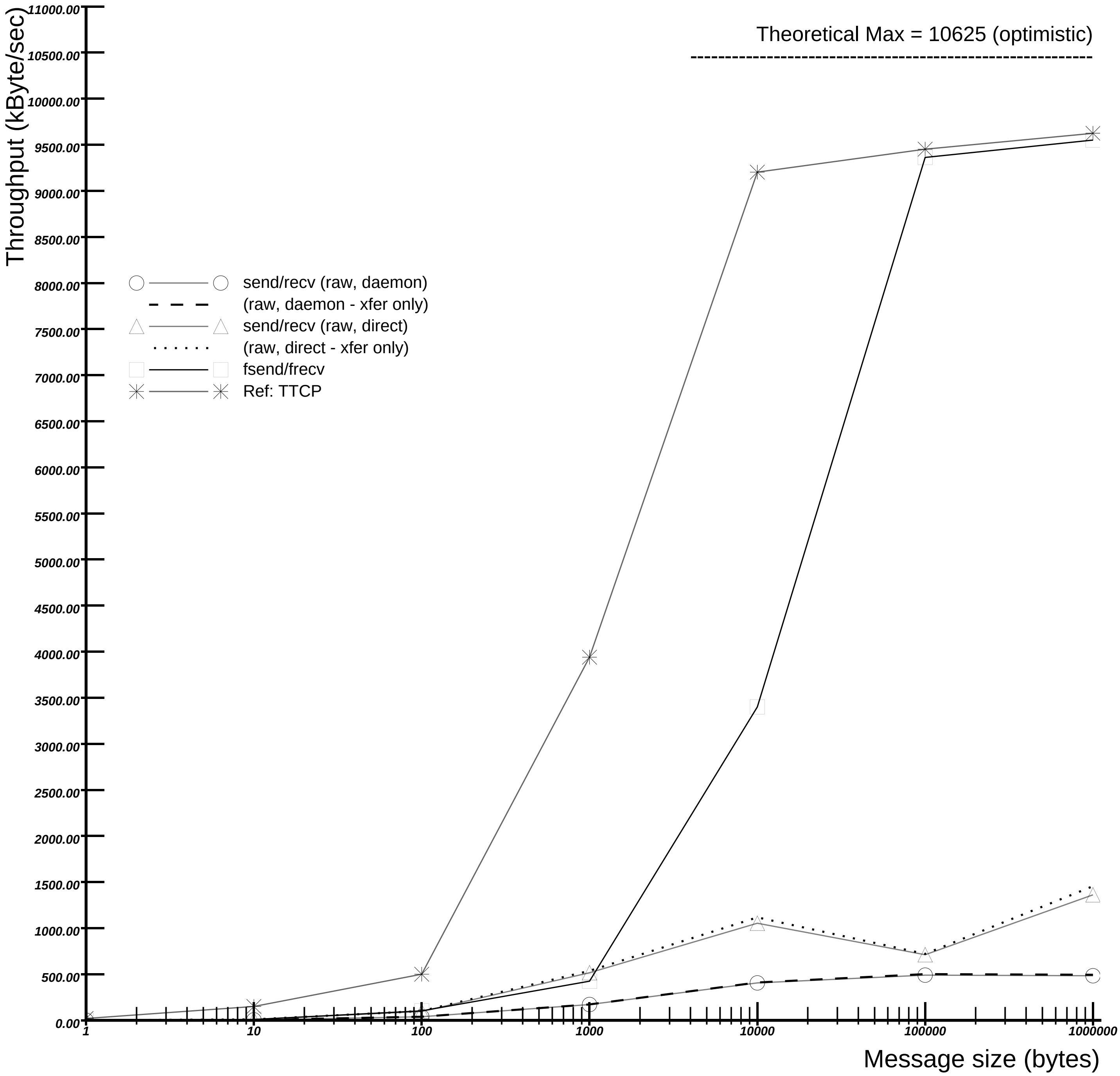
PVM Data Transfer Rates

– RS6000 on FDDI



PVM Data Transfer Rates (FDDI-IBM RS6000)

– SGI on Gigaswitch



PVM Data Transfer Rates (HEAT-SGI)

This page used for formatting only

- Revised Performance Results

– V-cycle MG on Enhanced PVM

Platform	Time (secs)	Comm. vol.	Comm. (secs)	No. of msgs	Idle time
16 SS1+ Enet	138* (198*)	96 MB	85 (154)	2704	48
8 RS6000 FDDI	110 (229)	192 MB	52 (162)	1808	30
8 SGI Gswitch	168 (264)	192 MB	81 (112)	1808	50
Cray Y-MP/1	22	N/A	N/A	N/A	N/A
i860/128	8.6	?	?	?	?

– Conjugate Gradient on Enhanced PVM

Platform	Time (secs)	Comm. vol.	Comm. (secs)	No. of msgs	Idle Time
16 SS1+ Enet	605 (701)	370 MB	404 (480)	37920	390
4 RS6000 FDDI	203 (285)	130 MB	100 (192)	7116	88
9 SGI Gswitch	108 (130)	250 MB	46 (101)	19756	42
Cray Y-MP/1	12	N/A	N/A	N/A	N/A
i860/128	8.6	?	?	?	?

– Integer Sort on Enhanced PVM

Platform	Time (secs)	Comm. vol.	Comm. (secs)	No. of msgs	Idle time
16 SS1+ Enet	398* (607*)	150 MB	375 (595)	10115	370
8 RS6000 FDDI	318 (674)	560 MB	171 (610)	2491	129
8 SGI Gswitch	258 (770)	560 MB	140 (720)	2491	105
Cray Y-MP/1	11	N/A	N/A	N/A	N/A
i860/32	26	?	?	?	?
i860/64	17	?	?	?	?
i860/128	14	?	?	?	?

– 3-D Fast Fourier Transform on Enhanced PVM

Platform	Time (secs)	Comm. volume	Comm. (secs)	No. of msgs	Idle time
16 SS1+ Enet	520* (717*)	420 MB	385 (502)	3450	266
8 RS6000 FDDI	412 (645)	1500 MB	220 (395)	826	112
8 SGI Gswitch	228 (1070)	1500 MB	130 (516)	826	34
Cray Y-MP/1	29	N/A	N/A	N/A	N/A
i860/64	21	?	?	?	?
i860/128	10	?	?	?	?

PVM Futures

- Threads-based Network Concurrent Computing
  - Multi-threaded architecture; LWP's are the units of parallelism
  - Rationale and Motivations
    + Trends in WS, servers : small-scale SMMP's with 1-8 processors
    + Potential for smaller granularity; more natural problem decomposition
    + Little or no overhead for local inter-thread communication
    + Greatly increased potential for latency hiding, overlap of communication and communication
  - Computing 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

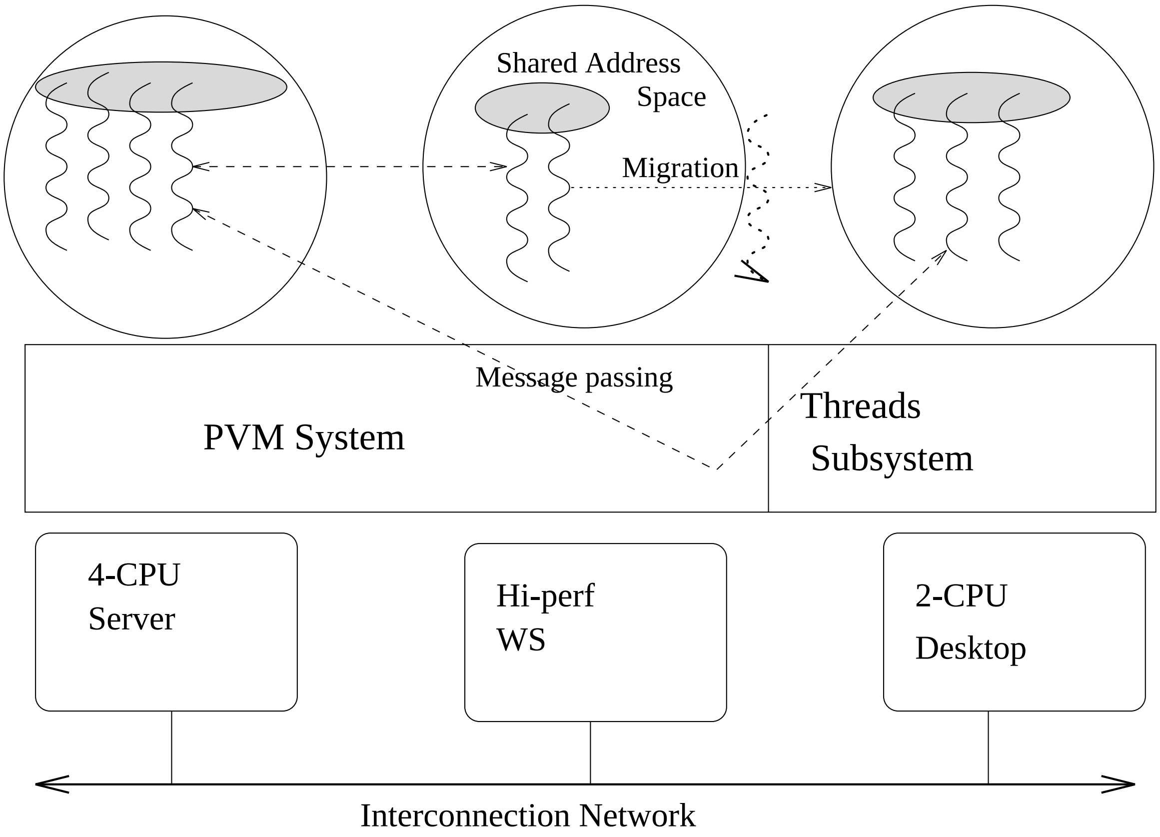
– Implementation

  + Based on minimal threads package — incorporating pre-emptive scheduling

  + Evolution of threads management primitives and design of system support

  + Thread interaction : transparent use of SM when local; explicit message passing when remote

- Threads-based Computing Model

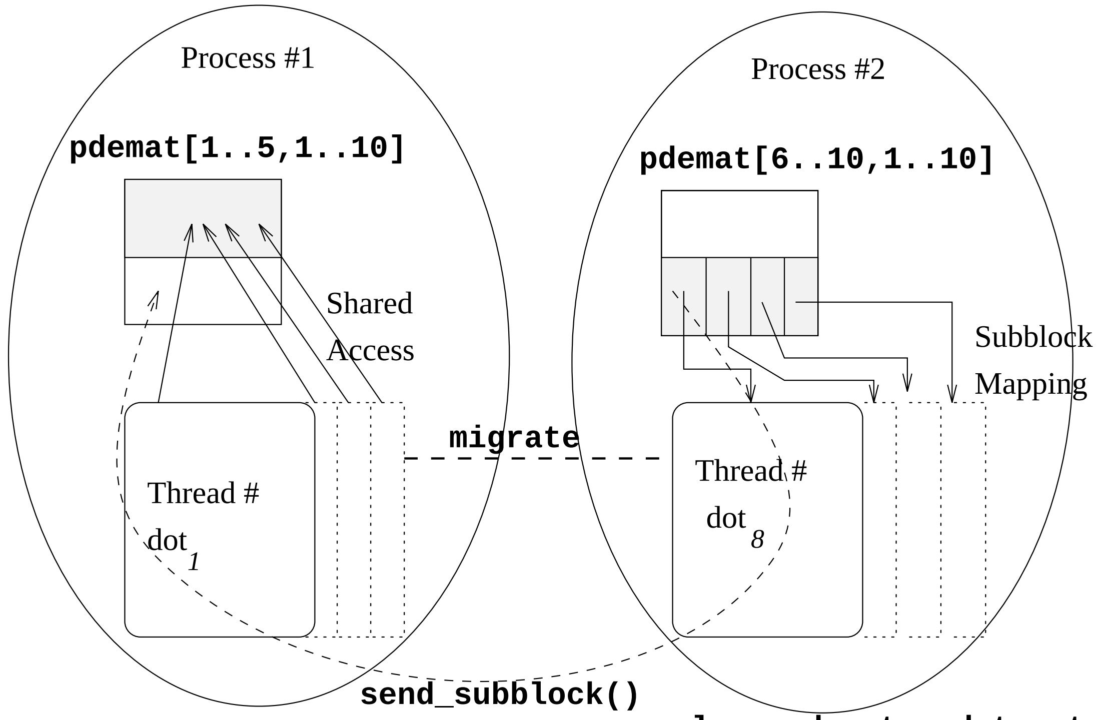


This page used for formatting purposes only

- Distributed Object Threads
  - Combine OO methodology with multithreading
  - Motivations
    + Usual advantages of OO development
    + Encapsulation and hiding of explicit communication; more efficient implementation
    + First class objects can be realized as threads, exploiting the advantages of the LWP framework
    + Potential for (semi-)automatic, dynamic, partitioning and workload allocation
  - DOT Computing Model
    + User specifies object class, deriving from one or more built-in base classes
    + Objects defined at fine granularity, e.g. one grid point, molecule, etc; or small sub-domains thereof
    + Aggregation objects/thechniques to construct variable sized collections - partly at compile time, partly dynamic
    + Object interactions described assuming shared address space at unit level, and message passing at aggregate level

+ Run time: aggregates built according to problem parameters; system partitions dynamically at aggregate level

- DOT Model Example



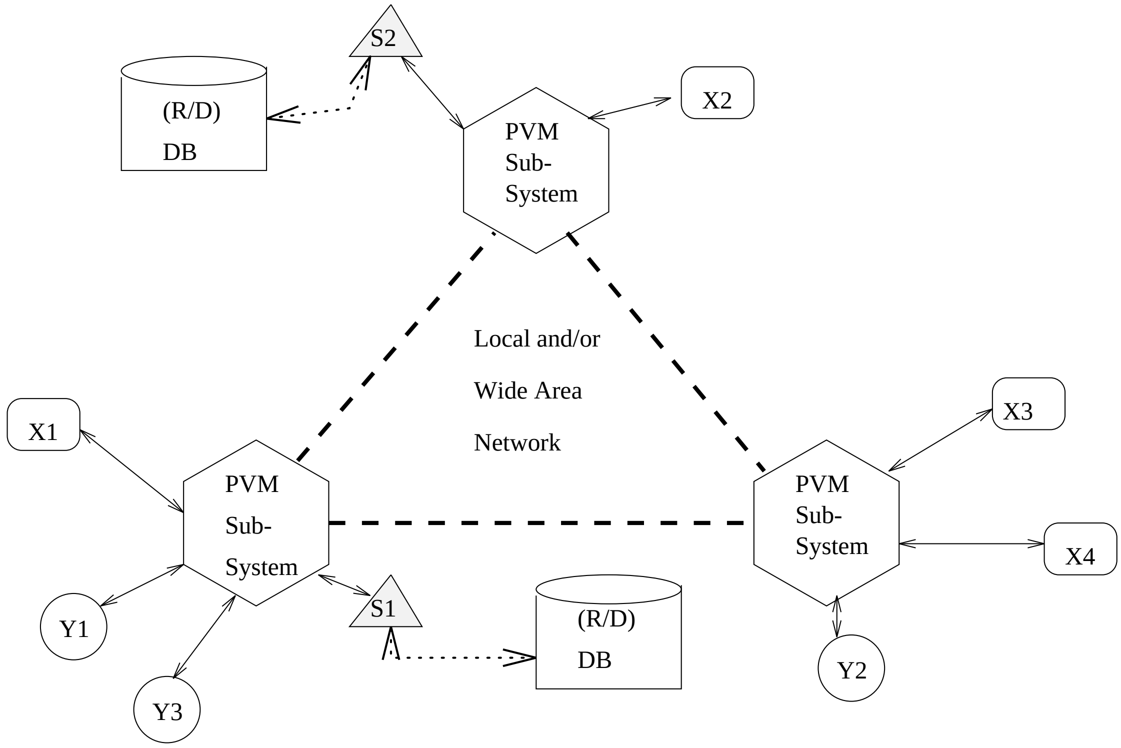

```
class dot_mat {
  float **elem;
  . . .
public:
  send_subblock();
  migrate_dot_mat();
  . . .
}
```

```
class pdemat : dot_mat {
  // private data
  . . .
public:
  jacobi_update();
  convergence_test();
  . . .
}
```

This page used for formatting purposes only

- The GDC Layer
  - Support for generalized distributed computing — I/O, groups, transactions
  - Rationale and Motivations
    + Parallel I/O infrastructure for real-world concurrent computing
    + Group facilities for large, multi-algorithm applications
    + Database and other applications: transaction and recovery facilities
  - Infrastructural requirements
    + Parallel I/O
    + Locking and synchronization
    + Data replication, fault tolerance
    + Client/Server paradigm
    + Transaction processing support

- GDC Architectural Model



X* "X" Session participants (e.g. CFD Computation)

Y* "Y" Session participants (e.g. Graphics filters/Visualization)

S* Distributed data management and I/O

Architectural Overview of GDC Layer

This page used for formatting purposes only

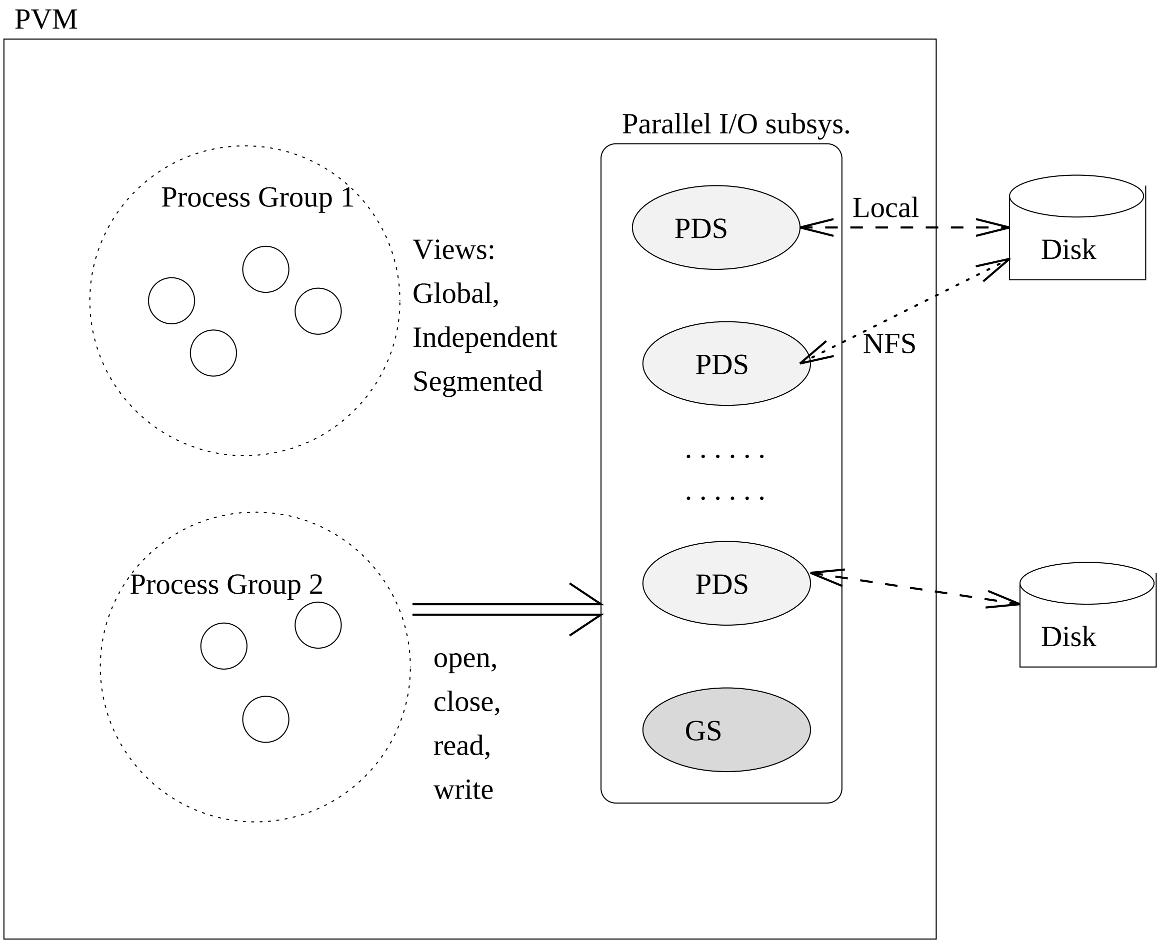
- Sessions and Contexts
  - Multiple, possibly intersecting, sessions or groups
    - + Different consistency and FT semantics as appropriate
    - + **`pvm_checkin("session_name",`**
      <auth params>, *<semantics>*, **`. . .);`**
    - + Returns *session_id* and *participant_id*.
    - + A process may check into multiple sessions
    - + **`pvm_checkout(`***<session_id>***`);`**
- Parallel I/O in PVM
  - Extension of Unix filesystem semantics
  - Different levels of shadowing, consistency semantics
  - Basic operations
    - + **`pvm_open("file[@machine]",`** *<options>***`);`**
    - + Options: conversion, compression, encryption, shadowing, access mode
    - + Returns *pvm_fd* handle for subsequent IO.
    - + **`pvm_write(pvm_fd, dp, n, . .);`**
    - + **`pvm_read(pvm_fd, dp, nitems, . .);`**

    + Atomic **`seek`** incorporated in read/write.
    + **`pvm_close(pvm_fd,`** *<flags>* **`);`**
    + Flags for IO include timeout component
    + Shadowing: Multiple copies of parallel file; replicated operations; strong or weak ("best effort").
    + Serialization: automatic, depending on modes.

- Locking and Synchronization
  - Abstract locking facility, with user-named locks.
  - System support for "persistence", deadlock avoidance, etc.
  - Locks (and resources) named by *<string, number>* tuple.
    + **`pvm_lock("lockname", locknum, flags);`**
    + Safety and timeout options provided.
    + Example convention for record-level locking:

      `pvm_lock("filename", recnum, AUTO);`

- Parallel I/O Architecture



This page used for formatting purposes only

- Client/Server Infrastructure
  - Application-specific services for distributed access
  - Location transparency, data conversion, replicated and/or shadow services for load balancing/failure resilience.
  - Simplified client and server development:
    + **`pvm_export("service",`** *<procname>*, *<msgtype* | *arglist>*, **`flags);`**
    + **`pvm_invoke("service", dataptr, TYPE | NARGS, flags);`**

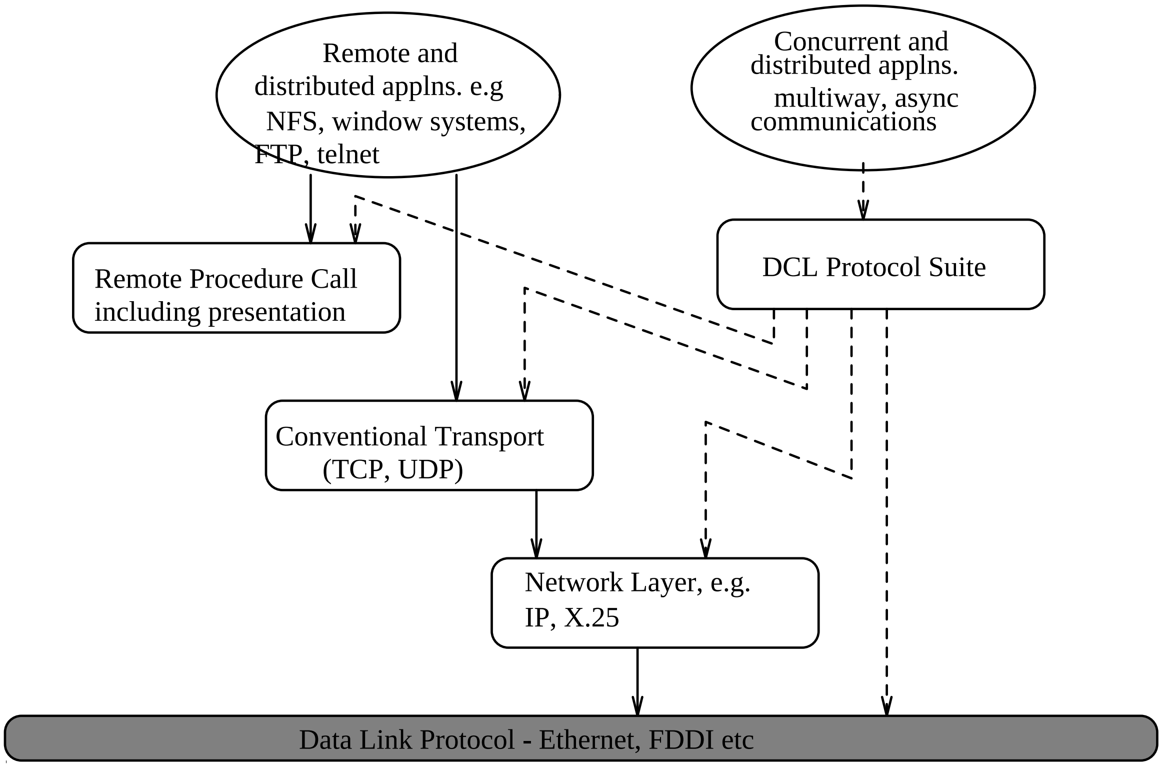
- Transaction Processing Support
  - Provide qualified ACID properties
  - Performance and response dependent on nesting and conflicting accesses
  - Ideally suited for "shared nothing" databases, high degree of inter-transaction parallelism.
  - Inter-DML parallelism may be implemented using PVM subsystem and other OS facilities (e.g. threads).

- Straightforward, concise, interface:
  - +
```
pvm_bt("trans name", number);
          . . .
          pvm_bt("nested", num);
            . . .
          pvm_et();
          if (. . .) pvm_abort();
pvm_et();
```
- Support for recovery, temporary rollback, logging.
- Semantics for message passing within transaction scope are presently unresolved.

● Implementation strategies for GDC

- Files: Combination of NFS, message-based IO schemes.
- Other Facilities:
  - + Distributed mutual exclusion (FPP, quorum)
  - + Distributed state-maintenance algorithms
  - + Fast multicast/broadcast & synchronization protocols

- Alternative Protocol Architectures
  - Optimizing message passing
    - + Reduced "within-host" protool processing
    - + Direct access to local network interfaces
  - Experimental DCL protocol
    - + Latency improvement : 45% - 50%
    - + Throughput improvement : 30% - 40 %
    - + Broadcasting : 16 way at 2x point-to-point



This page used for formatting purposes only

Concluding Remarks

- Heterogeneous Concurrent Computing
  - Viable technology for high performance computing
  - Good to excellent 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.
  - Management and administration - batch scheduling, queueing systems.