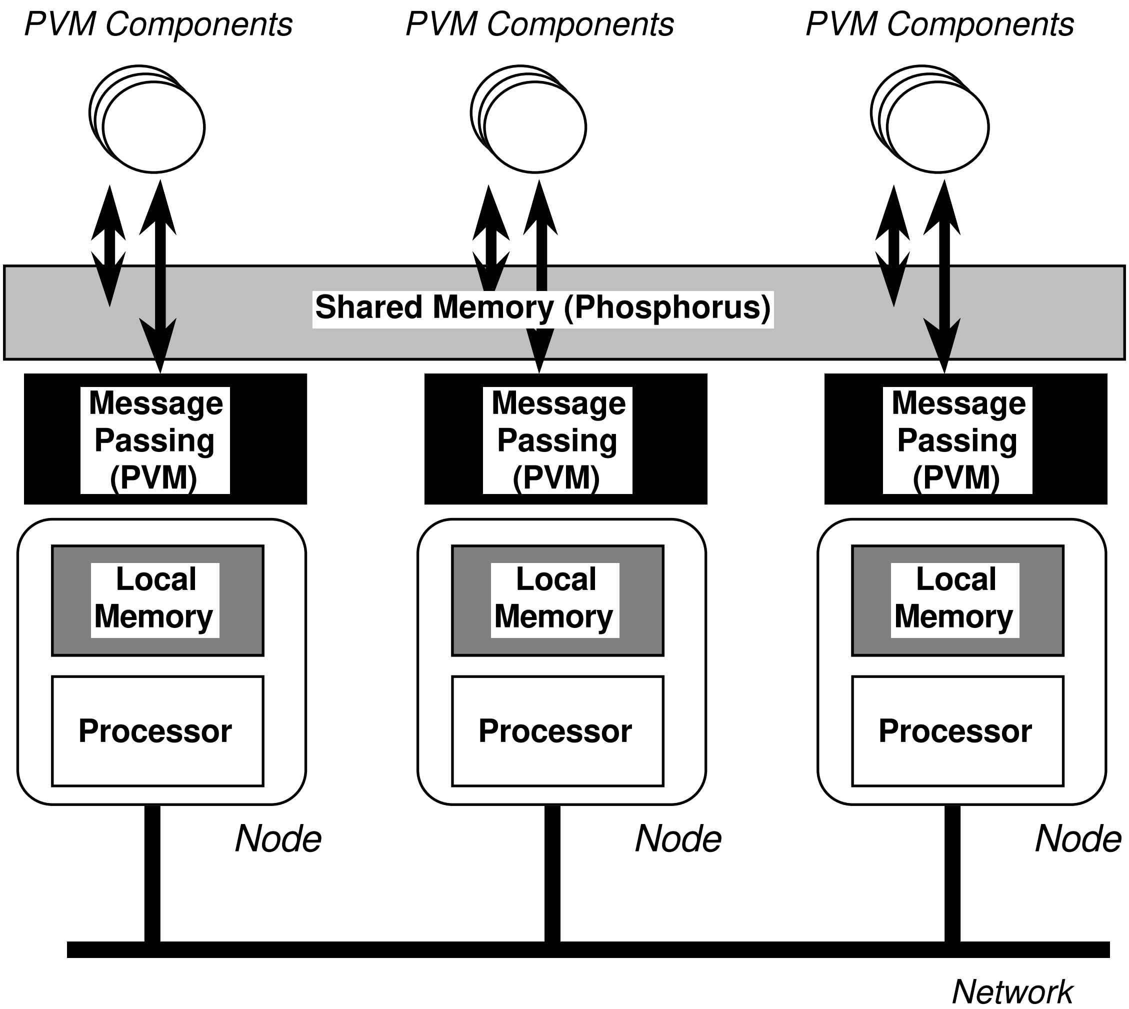
Phosphorus: Adding Shared Memory to PVM

Rocio Cabrera-Dantart
Isabelle Demeure
Philippe Meunier

Télécom Paris
Computer Science Department

The phosphorus is a mineral element that is said to play an important role in the performance of the human memory.

Shared Memory on Top of PVM



Goals and Design Choices

- Goals:
  - Provide PVM users with shared variables (in addition to message-passing).
  - Be as close as possible to the PVM philosophy.
  - Good performance.
  - Portability.
  - Evolution natural with PVM evolutions (if backward compatibility exists).

- Design choices:
  - The unit of sharing is the variable (not the page).
  - We support the types supported by the pack/unpack functions.
  - Support a variety of protocols and consistency models (similar to those supported by Munin).
  - Shared memory management is dynamic and distributed.
  - Implementation is done on top of PVM.

Phosphorus Primitives

- Process Control
  - int phos_init(void)
  - int phos_end(void)

- Variable Declaration
  - int phos_declare(int desc, int type, int count, int protocol)

    type (PVM types): byte, cplx, dcplx, double, float, int, long, short, uint, ulong, ushort.

    protocol : READ_ONLY, CONVENTIONAL, MIGRATORY, WRITE_SHARED and SYNCH.
  - int phos_free(int desc)

- Access to Variables
  - int phos_read(int desc, void *buffer)
  - int phos_write(int desc, void *buffer)

- Synchronization
  - int phos_lock(int desc)
  - int phos_unlock(int desc)
  - int phos_barrier(char *group, int count)

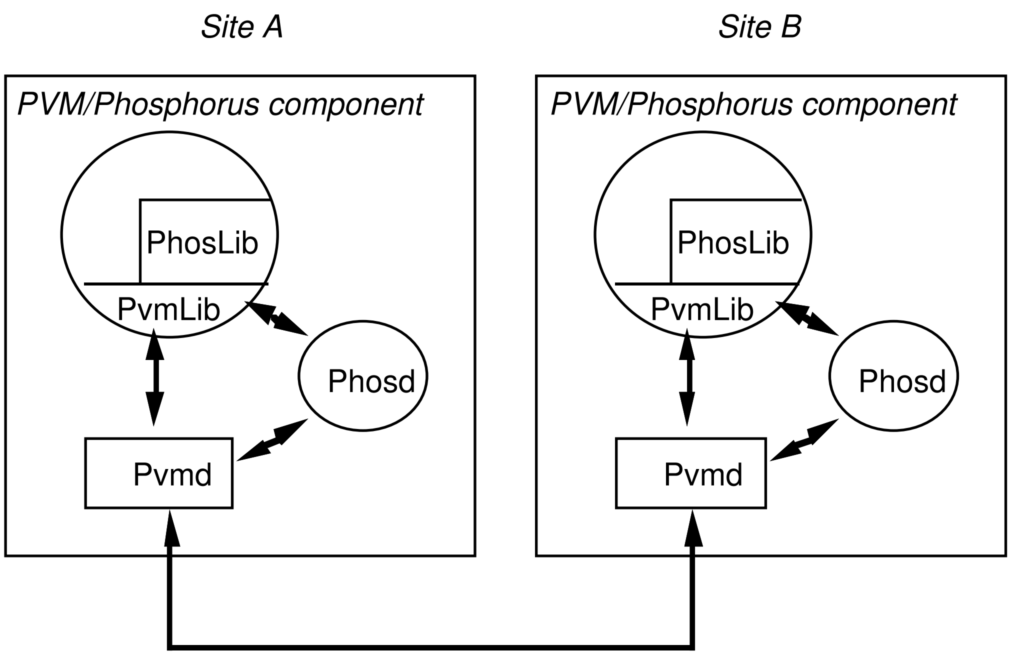
Example with PVM

MASTER	SLAVE
pvm_initsend();	
pvm_pk*();	
for(i=0; i<n_slaves; i++)	
pvm_send();	
	pvm_recv();
	pvm_upk*();
	compute(...);
	pvm_initsend();
	pvm_pk*();
	pvm_send();
for(i=0; i<n_slaves; i++)	
{	
pvm_recv();	
pvm_upk*();	
}	

With Phosphorus

MASTER	SLAVE
phos_write();	
phos_barrier();	
	phos_barrier();
	phos_read();
	compute(...);
	phos_write();
	phos_barrier();
phos_barrier();	
phos_read();	

Implementation



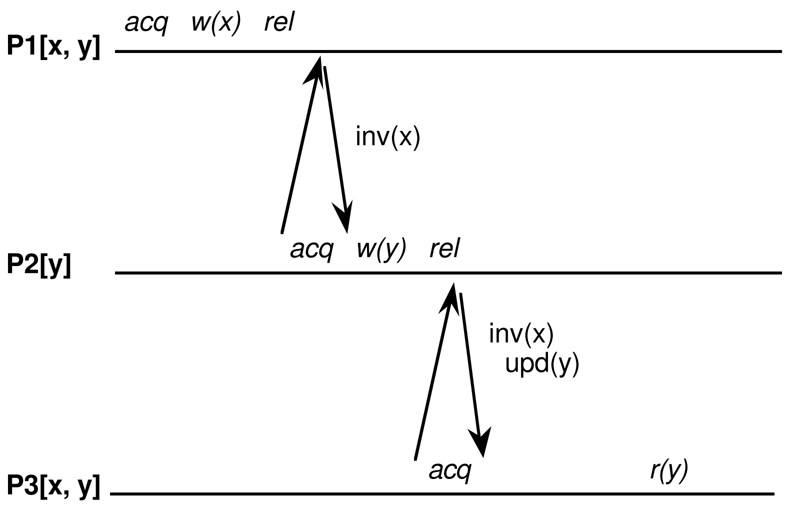
Phosphorus protocols

- Following Zwaenepoel and Carter (MUNIN) :

	Non-replicated	Replicated
Non-migrating	Central	Full replication ***Write-shared*** (MRMW) & ***Read-only***
Migrating	Migration ***Migratory*** (SRSW)	Read replication ***Conventional*** (MRSW)

Consistency protocols

- Coherence: general term for the semantics of memory operations.
- Consistency: refers to a specific kind of memory coherence [Nit91].
- Migratory / Conventional: sequential consistency (operations done in some sequential order).
- Write-shared: Lazy Hybrid Release Consistency.
- Release consistency :
  - multiple writers protocol.
  - the effects of shared memory accesses are delayed until synchronisation accesses (acquire and release) occur.
  - e.g. processes can simultaneously write to independant portions of their copy of a shared variable. A subsequent merge takes place at a synchronisation operation (lock and barrier).
  - eager invalidate / eager update / lazy invalidate / lazy update / lazy hybrid.
  - in Phosphorus: Lazy Hybrid Release Consistency.



Other DSM systems

- Ivy: Li & Hudak
  - One of the first DSM systems (on Apollo workstations).
  - A choice of algorithms to manage coherence (central, static distributed, dynamic distributed).
  - Unit of sharing = page.

- Munin: Carter & Zwaenepoel.
  - On SUN workstations.
  - Several protocols (central, full replication, migration, read replication).
  - Release consistency protocol (earger update).

- LINDA: Gelertner et al
  - tuple space.
  - unique id (key).
  - primitives: in, out, read, write, eval.

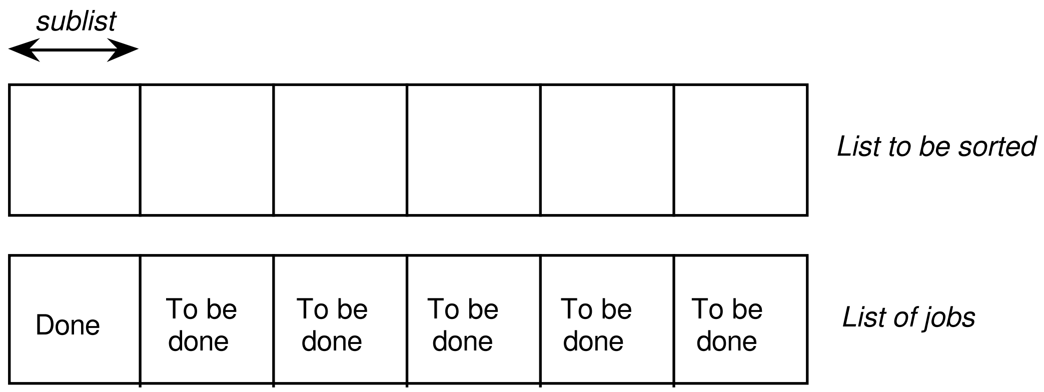
DSM Around PVM

- GLENDA: LINDA on top of PVM
    - takes PVM users in another domain (different philosophy).

- ADSMITH (A DSM in Tsing Hua university, June 1994):
    - many good ideas.
    - partly rewrites the set of PVM primitives.
    - implementation : 3 levels of memory (including a process cache) instead of 2 in Phosphorus.
    - fixed ownership of variables may cause more trafic.
    - performance ?

- PVM shmem:
    - in an earlier version of PVM (could be thought of as an earlier version of Phosphorus).
    - allocation/deallocation of shared memory segments.
    - synchronizations through locks.
    - dropped for performance reasons ?

Merge Sort



Shared Variables

- List to be sorted (in theory write-shared; in practice each sublist is migratory).
- List of jobs (migratory).
- Master performs initializations and merges the sorted sublists.
- workers pseudo-code :

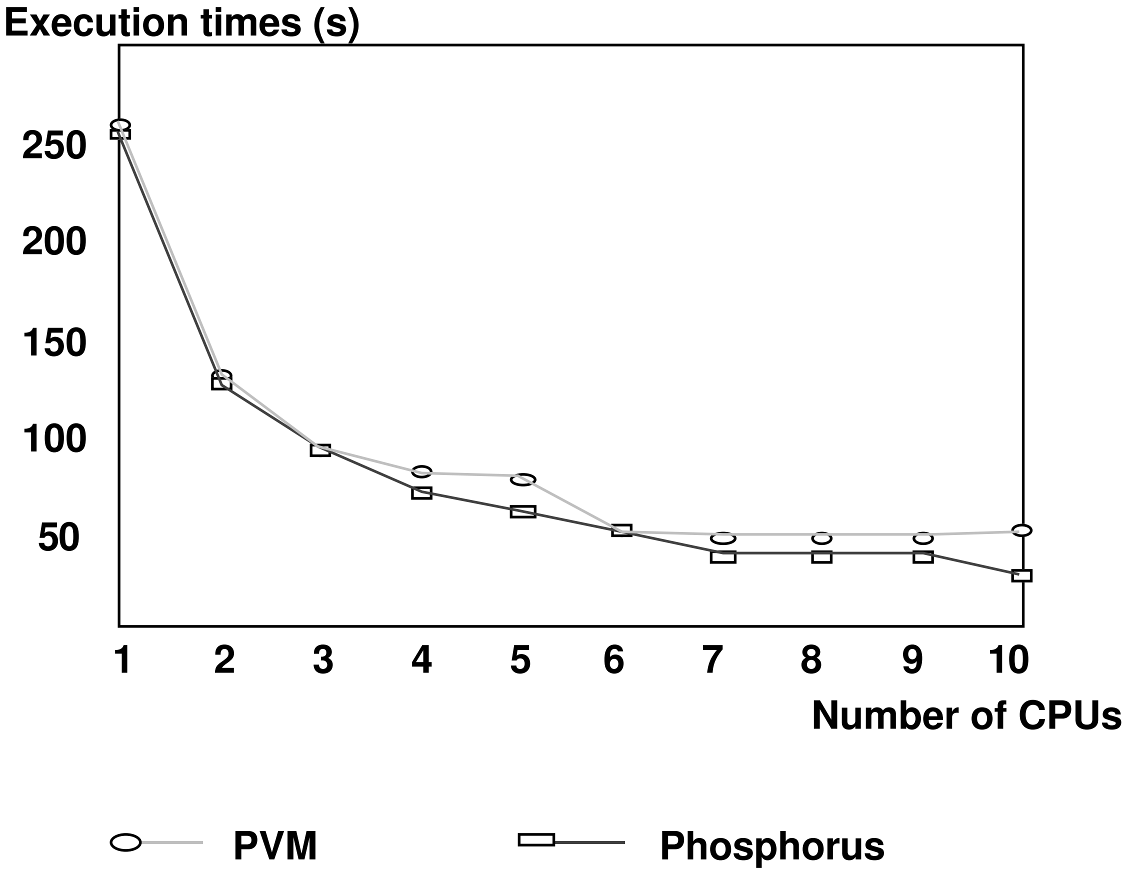
```
while (work-to-be-done)
   lock(job_list);
   find job to be done;
   if (no such job)
      work-to-be-done = false;
      unlock(job_list);
   else
      update job_list;
      unlock(job_list);
      read(sublist);
      sort(sublist);
      write(sublist);
phos_barrier;
```

Execution Times -Speed-up

Phosphorus versus PVM
1 list vs many sublists (migratory)

Phosphorus versus PVM Execution Times

(20 jobs)



Conclusions, Future Work

- 70% of the system has been prototyped in C on a network of SUN 4 workstation (7000 lignes of C code).
- The first performance tests are encouraging.
- We will continue the development and :
  - work on the support of shared structures.
  - tune the implementation for better performance.
  - evaluate several consistency models.
  - caracterise the sharing patterns that can benefit from the support of Phosphorus and the protocols adapted to these patterns.
  - bring the system to a point where it can be used by others (for teaching and research purposes).