Memory Coherence in Shared Virtual Memory Systems
Kai Li and Paul Hudak
PODC 1986

Why is Ivy cool?
  All the advantages of *very* expensive parallel hardware.
  On cheap network of workstations.
  No h/w modifications required!

Typical s/w arrangement:
  For parallel sort.
  Load array into memory.
  Each host responsible for a piece of the array.
  On host i:
    Do a local sort on its array piece.
    done[i] = true;
    Wait for all done[] to be true.
    Now merge my piece with neighbors...

Do we want a single address space?
  Or have programmer understand remote references?

Shall we make a fixed partition of address space?
  I.e. first megabyte on host 0, second megabyte on host 1, &c?
  And send all reads/writes to the correct host?
    We can detect reads and writes using VM hardware.
    I.e. I read- or write-protect remote pages.
  What if we didn't do a good job of placing the pages on hosts?
  Maybe we cannot predict which hosts will use which pages?

Could move the page each time it is used.
  When I read or write, I find current owner, and I take the page.
  So need a more dynamic way to find current location of the page.
  What if lots of people read a page?

Move page for writing, but allow read-only copies.
  When I write a page, I invalidate r/o cached copies.
  When I read a non-cached page, I find most recent writer.
  Works if pages are r/o and shared, or r/w by one host.
  Only bad case is write sharing.
  When might this arise? False sharing...

How does their scheme work? Section 3.1
  three CPUs, one MGR, draw table per entity

ptable: on per CPU, one entry per page
  lock
  access (read or write or nil)
  i_am_owner (same as write?)

info: just on MGR, one entry per page
  lock
  copy_set
  owner

Message types:
  RQ read query (reader to MGR)

```
  RF read forward (MGR to owner)
  RD read data (owner to reader)
  RC read confirm (reader to MGR)
  WQ
  IV
  IC
  WF
  WD
  WC

scenario 1: owned by CPU0, CPU1 wants to read
  0. page fault on CPU1, since page must have been marked invalid
  1. CPU1 sends RQ to MGR
  2. MGR sends RF to CPU0, MGR adds CPU1 to copy_set
  3. CPU0 sends RD to CPU1, CPU0 marks page as access=read
  4. CPU1 sends RC to MGR

scenario 2: owned by CPU0, CPU2 wants to write
  0. page fault on CPU2
  1. CPU2 sends WQ to MGR
  2. MGR sends IV to copy_set (i.e. CPU1)
  3. CPU1 sends IC msg to MGR (or does it? does MGR wait?)
  4. MGR sends WF to CPU0
  5. CPU0 sends WD to CPU2, clears access
  6. CPU2 sends WC to MGR

what if two CPUs want to write the same page at the same time?

problem:
  a write has many steps and modifies multiple tables
  the tables have invariants:
    MGR must agree w/ CPUs about the single owner
    MGR must agree w/ CPUs about the copy_set
    copy_set != {} must agree with owner's writeability
  thus write implementation should be atomic

what enforces the atomicity?

what are the RC and WC messages for?
  what if RF is overtaken by WF?
  (I'm not sure about this. Perhaps they assume FIFO message order
anyway?)
  (per-CPU ptable locks fix many potential races.)

does Ivy provide strict consistency?
  no: ST may take a long time to revoke read access on other CPUs
  so LDs may get old data long after the ST issues

does Ivy provide sequential consistency?
  does it work on our examples?
  v = fn(); done = true;
  can a cpu see done=true but still see old v?

Ivy does seem to use our two seq consist implementation rules.
  1. Each CPU to execute reads/writes in program order, one at a time
  2. Each memory location to execute reads/writes in arrival order, one
at a time
```

```
What's a block odd-even based merge-split algorithm?
  Why is appropriate to this kind of parallel architecture?
  Partition over N CPUs.
  Local sort at each CPU.
  View CPUs as logical chain.
  Repeat N times:
    Even CPUs send to (higher) odd CPUs.
    Odd CPUs merge, send low half back to even CPU.
    Odd CPUs send to (higher) even CPUs.
    Even CPUs merge, send low half back to odd CPU.
  Note that "send to" means look at the right place in shared memory.
    Probably everything in a single huge array.

What's the best we could hope for in terms of performance?
  Linear speedup...

When are you likely to get linear speedup?
  When there's not much sharing -- strict partition.

How well do they do?
  linear for PDE and matrix mul, not so good for sorting

In what sense does it subsume the notion of RPC?
  When would DSM be better than RPC?
    More transparent. Easier to program.
  When would RPC be better?
    Isolation. Control over communication. Tolerate latency.
    Portability.
  Might you still want RPC in your DSM system? For efficient
sleep/wakeup?

Has their idea been successful?
  Using workstations for parallel processing: yes! Beowulf, Google, &c.
  Shared memory? Hard to say. Lack of control over communication
details?
```