Appendix A. Message-passing algorithms

A.1 Paradigms

There are many different message types used, but they fall in roughly 3 categories:

Global sums, data updates and flow control.

Global sum:

Each node computes local value and sends it to node 0 for global summation.

Data update:

Each node sends a request of which data it needs from another node, then receives the data.

Flow control:

Node 0 collects done messages from all nodes, then broadcasts “go-on” message to all of them.

Note: in this way it is possible to do without barrier synchronization calls, since the J machine did not have them. In effect we implement our own barriers.

A.2 Pseudo-code

Main
   Read_ctl
   Setup_ee
   Master_cmm
}

master_cmm
   if not nbs setup calls dyn()
   else proceeds to update multipoles.

Compute multipoles

Dyn() {
   Self_far_nearinf;
}
Roundup_near_atom;
Unpack_near_atoms;

Calculate exclusions, torsion… no communication

Send_partial_forces home

Integrate1:
Compute KE
Integrate 2;

Handle_moving;

Compute farfields: is:

Calc_center_for_all_cells;
  Checkwindow cald, which :
      Combine_calc_center for own
      Send message CALC type for others.
      At end broadcasts all_done_calc_type

  Proceeds to :
      Process CLACL_TYPE, via combine_calc_center
      ALLDONECLACTYPE, by conter., via which exits.

Check_window child _pnc
  Compute multipoles or call child if can,
  Send CHILD_TYPE if not own
  Then do update taylor of PNC_M-TYPE if not own
  At end broadcasts ALL_DONE_CALC_TYPE

Case:
- child
- master_pnc
  -- done by counter.

Check_window parent
  Combine taylor if own
  Send parent type otherwise
  ALL_DONE_CALC at end is broadcast
Process PARENT via parent
  ALL DONE CALC via counter.
  At end sends DONE CMM to 0
  0 counts them and broadcasts ALL_DONE_CMM

AT ALL DONE CMM, dyn is called.

Self_far_nearinfl:
Sends DONE_CMM

Processing loop (while done):

- ATOMS_M: master_atoms
- DONE_CMM_TYPE:
- Farinfl;selfinfl – no communicatino in those
  And sends ack_near_type to start nearinfl processing.
- DONE_SELF_FAR_TYPE:
- Counter, at counter finished broadcast MOVE_ON_TYPE

  MOVE_ON_TYPE: sets done flag
  Ack_NEAR_TYPE: check_window_ack_near

At exit sends DONE_SELF_FAR_TYPE

Check_window_ack_near

At exit sends DONE_SELF_FAR_TYPE

Otherwise does atoms if own
Or sends ATOMS_M_TYPE if nonlocal

Roundup_near_atoms

  init_DestCounter(void) /* zeroed DesBuf and DestCounter.
  get_near_atoms(SEND_FORCES_TYPE,MOVE_ON_TYPE);
  Counter = Numnodes-1
  while (Counter) {
      crecv(SEND_FORCES_TYPE,
      expects only SEND FORCES TYPE messages and exactly one from a node.
      memcpy(CellBuf[infonode()],NearMsg,(int)(sizeof(NEAR)*Size));
      dumps them directly into CellBuf with offset of the one that are received.
  }
  get_near_atoms(int send_type,int done_type)
  for all leaf cells on this CPU (iterated through leafhead and ->lnext
  iterate through 27 neighbours.
  If neighbour does not reside on our CPU
  add_cell_to_buf(n,who_i_am,Dest), which does:
  DestCounter[dest]++;
  DestBuf[dest]++;
  And puts the cell number to
  DestBuf[dest]-><dest_cell

For each cpu I, which is not ours {
  (outstanding is zero here)
  outstanding = outstanding + DestCounter[i]; destcounter
  thus outstanding shows ;
how many we are sending to this I cpu;

I
    f (i != who_i_am) {
        Csend(send_type, HeadDestBuf[i], sizeof(NEAR)*(DestCounter[i]+1), i, 0);
    } // if this is the basic send.

    DestCounter[i]=0; /* this is the basic send.

}  // if we did not need to send any to other nodes:

send done_type with the number outstanding in the message.

Unpack_near_atoms ()

For all other CPUS.

    send_atoms(Index); /* this generates lots of UNPACK_FORCE_TYPE messages*/

while (Counter != 0) {
    crecv(-1, RcvMsg, sizeof(DYN_MSG));

switch processes:

    case UNPACK_FORCE_TYPE:
        casts DYN_MSG to type NEAR_MSG
        looks up atom in local hash
        if not updated. updates f,v,x, and where it came from,
        sets flag updated to 1. (who clears it);
        NEAR_MSG has another field about bond data, which is
        updated into nbe arrays. (probably does not need that ?)

    case MOVE_ON_TYPE:
        Counter--;
        break;
}

send_atoms(int Index);
void send_atoms(int Index)
{
    double TempBuf;

    printf ("%d: send_atoms called.
", who_i_am);

    while (CellBuf[Index]->dest_cell != NULL) {
        pack_atoms(CellBuf[Index], Index, 0, UNPACK_FORCE_TYPE, 0);
        CellBuf[Index]++;
    }

    if (Numnodes != 1)
        csend(MOVE_ON_TYPE, &TempBuf, sizeof(double), -1, 0);
    csend(MOVE_ON_TYPE, &TempBuf, sizeof(double), who_i_am, 0);
}

void send_partial_forces_home(void)
{
    for all atoms in localhashable (processed via atomlist[I] and lookup_atom_in hash and iterator ->hnext
    memcpy(ForceMsgMin->atom, A, sizeof(ATOM)); /* this is one atom */
    csend(UNPACK_PARTIAL_TYPE, ForceMsgMin, sizeof(NEAR_MSG_MIN), A->sen
der,0); outstanding++;

if nothing ever was send then do csend(PARTIAL_DONE_TYPE,PartialMsg,sizeof(ATOM),0,0)
else start receiving:
  case UNPACK_PARTIAL_TYPE:
    update forces in hash and force_updated flag and
    csend(ACK_UNPACK_PARTIAL_TYPE,&i,
  case ACK_UNPACK_PARTIAL_TYPE:
    • outstanding--; / decrease counter;
    • if counter is zero, csend(PARTIAL_DONE_TYPE,Pa

  case PARTIAL_DONE_TYPE:
    adds the message to the global PE (that’s how it is send actually)b
    when PARTIAL_DONE_TYPE received from all nodes,
    broadcast PE back to all of them via MOVE_ON_TYPE
  case MOVE_ON_TYPE:
    receive the new global_pe and set done to 0, so receiving loop terminates.

Integrate()

Integrate->begin
  Calc_ke calculates local ke and sends it to 0 via csend(ENERGIES_TYPE

Collect_kes
  On node zero: Receives ENERGIES_TYPE and adds them together then broadcast via
  GLOBAL_ENERGIES_TYPE
  On ondes other than zero: recv GLOBAL_ENERGIES_TYPE.

Integrate_force2()
  Integrate->force2()
Sends MOVE_ON_TYPE to 0
Then zero receives it from all,
Adds up global kinetic energy.
Broadcasts INTEGRATE_VEL_TYPE on 0 and receives it on the other ones.