Hybrid MPI+OpenMP Parallel MD

Aiichiro Nakano

Collaboratory for Advanced Computing & Simulations
Department of Computer Science
Department of Physics & Astronomy
Department of Chemical Engineering & Materials Science Science
University of Southern California

Email: anakano@usc.edu
Hybrid MPI+OpenMP Programming

Each MPI process spawns multiple OpenMP threads

In a PBS script:
```
mpirun -np 2
```

In the code:
```
omp_set_num_threads(3);
```
MPI+OpenMP Calculation of $\pi$

- Each MPI process integrates over a range of width $1/n\text{proc}$, as a discrete sum of $n\text{bin}$ bins each of width $\text{step}$
- Within each MPI process, $n\text{threads}$ OpenMP threads perform part of the sum as in $\text{omp\_pi\_c}$
#include <stdio.h>
#include <mpi.h>
#include <omp.h>
#define NBIN 100000
#define MAX_THREADS 8

void main(int argc,char **argv) {
    int nbin,myid,nproc,nthreads,tid;
    double step,sum[MAX_THREADS]={0.0},pi=0.0,pig;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Comm_size(MPI_COMM_WORLD,&nproc);
    nbin = NBIN/nproc; step = 1.0/(nbin*nproc);
    omp_set_num_threads(2);
    #pragma omp parallel private(tid)
    {
        int i;
        double x;
        nthreads = omp_get_num_threads();
        tid = omp_get_thread_num();
        for (i=nbin*myid+tid; i<nbin*(myid+1); i+=nthreads) {
            x = (i+0.5)*step; sum[tid] += 4.0/(1.0+x*x);
            printf("rank tid sum = %d %d %e\n",myid,tid,sum[tid]);
        }
        for (tid=0; tid<nthreads; tid++) pi += sum[tid]*step;
        MPI_Allreduce(&pi,&pig,1,MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD);
        if (myid==0) printf("PI = %f\n",pig);
    }
    MPI_Finalize();
}
MPI+OpenMP Example: \texttt{hpi.c}

- **Compilation on hpc-login2.usc.edu**
  
  source /usr/usc/mpich/default/mx-intel/setup.csh
  
  mpicc -o hpi hpi.c -openmp

- **PBS script**
  
  ```bash
  #!/bin/bash
  #PBS -l nodes=2:ppn=1,arch=x86_64
  #PBS -l walltime=00:00:59
  #PBS -o hpi.out
  #PBS -j oe
  #PBS -N hpi
  WORK_HOME=/auto/rcf-proj2/an/anakano/hpc/cs596
  cd $WORK_HOME
  np=$(cat $PBS_NODEFILE | wc -l)
  mpirun -np $np -machinefile $PBS_NODEFILE ./hpi
  ```

- **Output**
  
  ```
  rank tid sum = 1 1  6.434981e+04
  rank tid sum = 1 0  6.435041e+04
  rank tid sum = 0 0  9.272972e+04
  rank tid sum = 0 1  9.272932e+04
  PI = 3.141593
  ```
OpenMP threads handle blocks of linked-list cells in each MPI process (= spatial-decomposition subsystem)
Linked-List Cell Block

Variables

- `vthrd[0|1|2]` = # of OpenMP threads per MPI process in the x|y|z direction.
- `thbk[3]`: `thbk[0|1|2]` is the # of linked-list cells in the x|y|z direction that each thread is assigned.

In `main()`:
omp_set_num_threads(nthrd);

In `init_params()`:
/* Compute the # of cells for linked-list cells */
for (a=0; a<3; a++)
    lc[a] = al[a]/RCUT; /* Cell size ≥ potential cutoff */
/* Size of cell block that each thread is assigned */
for (a=0; a<3; a++)
    thbk[a] = lc[a]/vthrd[a];
/* # of cells = integer multiple of the # of threads */
for (a=0; a<3; a++) {  
    lc[a] = thbk[a]×vthrd[a]; /* Adjust # of cells/MPI process */
    rc[a] = al[a]/lc[a]; /* Linked-list cell length */  
}
OpenMP Threads for Cell Blocks

Variables

- std = scalar thread index.
- vtd[3]: vtd[0 | 1 | 2] is the x|y|z element of vector thread index.
- mofst[3]: mofst[0 | 1 | 2] is the x|y|z offset cell index of cell-block.

```c
std = omp_get_thread_num();
vtd[0] = std/(vthrd[1]*vthrd[2]);
vtd[1] = (std/vthrd[2])%vthrd[1];
vtd[2] = std%vthrd[2];
for (a=0; a<3; a++)
    mofst[a] = vtd[a]*thbk[a];
```

Call `omp_get_thread_num()` within an OpenMP parallel block.
Threads Processing of Cell Blocks

- Start with the MPI parallel MD program, pmd.c
- Within each MPI process, parallelize the outer loops over central linked-list cells, \( mc[] \), in the force computation function, compute_accel(), using OpenMP threads
- If each thread needs separate copy of a variable (e.g., loop index \( mc[] \)), declare it as private in the OpenMP parallel block

```c
#pragma omp parallel private(mc,...)
{
    ...
    for (mc[0]=mofst[0]+1; mc[0]<=mofst[0]+thbk[0]; (mc[0])++)
    {
        Each thread handles \( thbk[0] \times thbk[1] \times thbk[2] \) cells independently
    }
    ...
}
```
Avoiding Critical Sections (1)

- Remove the critical section
  
  ```c
  if (bintra) lpe += vVal; else lpe += 0.5*vVal;
  ```
  
  by defining an array, `lpe_td[nthrd]`, where each array element stores the partial sum of the potential energy by a thread
Avoiding Critical Sections (2)

- To avoid multiple threads to access an identical force array element, stop using the Newton’s third law:

```c
int bintra;
...
if (i<j && rr<rrCut) {
    ...
    if (bintra) lpe += vVal; else lpe_td[std] += 0.5*vVal;
    for (a=0; a<3; a++) {
        f = fcVal*dr[a];
        ra[i][a] += f;
        if (bintra) ra[j][a] -= f;
    }
}
```

Mutually exclusive access to ra[][]!
1. Interactively submit a PBS job & wait until you are allocated nodes. (Note that you will be automatically logged in to one of the allocated nodes.)

$ qsub -I -l nodes=2:ppn=4,arch=x86_64 -l walltime=00:29:00
qsub: waiting for job 173932.hpc-pbs.usc.edu to start
qsub: job 173932.hpc-pbs.usc.edu ready

Begin PBS Prologue Thu Oct 27 08:24:22 PDT 2011
Job ID: 173932.hpc-pbs.usc.edu
Username: anakano
Group: m-csci
Name: STDIN
Queue: quick
Shared Access: no
Nodes: hpc1168 hpc1181
PVFS: /scratch (1.7T)
TMPDIR: /tmp/173932.hpc-pbs.usc.edu
End PBS Prologue Thu Oct 27 08:24:27 PDT 2011

[anakano@hpc1181 ~]$
Running HMD at HPC (2)

2. Type the following sequence of commands. (In this example, hpc/cs596 is my working directory, where the executable hmd is located.)

   [anakano@hpc1181 ~]$ bash
   bash-3.2$ cd hpc/cs596
   bash-3.2$ cp $PBS_NODEFILE nodefile

3. Edit nodefile, which originally consisted of 8 lines, to delete 6 lines.

   (Original nodefile)              (Edited nodefile)
   hpc1181                       hpc1181
   hpc1181                       hpc1168
   hpc1181
   hpc1168
   hpc1181
   hpc1168
   hpc1168
   hpc1168

4. Submit a two-process MPI program (named hmd); each of the MPI process will spawn 4 OpenMP threads.

   bash-3.2$ mpirun -np 2 -machinefile nodefile ./hmd
5. While the job is running, you can open another window & log in to both the nodes to check that all processors on each node are busy. Type ‘H’ to show individual threads.

[anakano@hpc-login2 ~]$ ssh hpc1181
[anakano@hpc1181 ~]$ top (then type H)

```
top - 08:27:03 up 21 days, 18:37, 1 user, load average: 0.32, 0.98, 2.50
Tasks: 337 total, 5 running, 332 sleeping, 0 stopped, 0 zombie
Cpu(s): 16.6%us, 0.1%sy, 0.0%ni, 83.3%id, 0.1%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 45319072k total, 1243820k used, 44075252k free, 507748k buffers
Swap: 1052216k total, 292k used, 1051924k free, 266928k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
15879 anakano   18   0 103m 7980 1792 R 99.7  0.0   0:06.22 hmd
15883 anakano   25   0 103m 7980 1792 R 99.7  0.0   0:06.20 hmd
15884 anakano   25   0 103m 7980 1792 R 99.7  0.0   0:06.19 hmd
15885 anakano   25   0 103m 7980 1792 R 99.4  0.0   0:06.18 hmd
15833 anakano   15   0 11020 1244  780 R  0.3  0.0   0:00.04 top
   1 root      15   0 10368  648 552 S  0.0  0.0   0:05.12 init
...```
• Instead of the interactive PBS job, you can also submit a batch job using the following script (the uniq command will eliminate duplicated lines in $PBS_NODEFILE and retain only one line per node).

```bash
#!/bin/bash
#PBS -l nodes=2:ppn=4,arch=x86_64
#PBS -l walltime=00:00:59
#PBS -o hmd.out
#PBS -j oe
#PBS -N hmd
WORK_HOME=/home/rcf-proj2/an/anakano/hpc/cs596
cd $WORK_HOME
cat $PBS_NODEFILE | uniq > nodefile
np=$(cat nodefile | wc -l)
mpirun -np $np -machinefile nodefile ./hmd
```

This way is recommended!
Strong Scalability of Hybrid MD

1 MPI process; 1-8 threads

Algorithmic improvement on Xeon