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
Department of Biological Sciences
University of Southern California

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

Each MPI process spawns multiple OpenMP threads

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

In the code:
```c
omp_set_num_threads(3);
```

- MPI processes communicate by sending/receiving messages
- OpenMP threads communicate by writing to/reading from shared variables
MPI+OpenMP Calculation of $\pi$

- **Spatial decomposition:** 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 $\text{step}$

- **Interleaving:** Within each MPI process, $n_{\text{threads}}$ OpenMP threads perform part of the sum as in `omp_pi.c`

\[
\pi = \int_0^1 \frac{4}{1 + x^2} \, dx = \Delta \sum_{i=0}^{N-1} \frac{4}{1 + x_i^2}
\]
#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: hpi.c

- **Compilation on hpc-login3.usc.edu**
  
  source /usr/usc/openmpi/default/setup.csh
  
  mpicc -o hpi hpi.c -fopenmp

- **PBS script**
  
  #!/bin/bash
  
  #PBS -l nodes=2:ppn=1
  #PBS -l walltime=00:00:59
  #PBS -o hpi.out
  #PBS -j oe
  #PBS -N hpi
  
  WORK_HOME=/auto/rcf-proj/anl/anakano
  
  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
Hybrid MPI+OpenMP Parallel MD

OpenMP threads handle blocks of linked-list cells in each MPI process (= spatial-decomposition subsystem)
Linked-List Cell Block

Variables

- \( \text{vthrd}_{[0|1|2]} \) = \# of OpenMP threads per MPI process in the x\( \mid \)y\( \mid \)z direction.
- \( \text{nthrd} \) = \# of OpenMP threads = \( \text{vthrd}_{[0]} \times \text{vthrd}_{[1]} \times \text{vthrd}_{[2]} \).
- \( \text{thbk}_{[3]} \): \( \text{thbk}_{[0|1|2]} \) is the \# of linked-list cells in the x\( \mid \)y\( \mid \)z direction that each thread is assigned.

In main():

```c
omp_set_num_threads(nthrd);
```

In init_params():

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

In hmd.h:

```c
int vthrd[3]={2,2,1},nthrd=4;
```
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 $\text{thbk[0]} \times \text{thbk[1]} \times \text{thbk[2]}$ 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 lpeTd[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 -l walltime=00:29:00
qsub: waiting for job 21631448.hpc-pbs2.hpcc.usc.edu to start
qsub: job 21631448.hpc-pbs2.hpcc.usc.edu ready
----------------------------------------
Begin PBS Prologue Wed Sep 28 09:59:08 PDT 2016
Job ID: 21631448.hpc-pbs2.hpcc.usc.edu
Username: anakano
Group: m-csci
Project: default
Name: STDIN
Queue: quick
Shared Access: no
All Cores: no
Has MIC: no
Nodes: hpc2566 hpc2568
Scratch is: /scratch
TMPDIR: /tmp/21631448.hpc-pbs2.hpcc.usc.edu
End PBS Prologue Wed Sep 28 09:59:19 PDT 2016
----------------------------------------
[anakano@hpc2566 ~]$ You are logged in to one of the allocated nodes
2. Type the following sequence of commands. (In this example, hpc/cs653 is soft link to my working directory, where the executable hmd is located.)

```
[anakano@hpc2566 ~]$ bash
bash-4.1$ cd hpc/cs653
bash-4.1$ cp $PBS_NODEFILE nodefile
```

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

```
(Original nodefile)                      (Edited nodefile)
hpc2566                                 hpc2566
hpc2566                                 hpc2568
hpc2566                                 →
hpc2566                                 hpc2568
hpc2568
hpc2568
hpc2568
hpc2568
```

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

```
bash-4.1$ mpirun ./hmd --bind-to none -np 2 --machinefile nodefile
```

Note: Without the “--bind-to none” option, OpenMPI would bind each process to a single core, hence all OpenMP threads would run on the same core.
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-login3 ~]$ ssh hpc2566
[anakano@hpc2566 ~]$ top (then type H)

```
top - 10:03:23 up 144 days, 17:40, 53 users,  load average: 16.14, 16.40, 16.14
Tasks: 1053 total,   4 running, 1045 sleeping,   1 stopped,   3 zombie
Cpu(s):  20.4%us,  8.0%sy,  0.0%ni,  62.3%id,  9.1%wa,  0.0%hi,  0.1%si,  0.0%st
Mem:  65931952k total, 65498968k used,  432984k free,  331476k buffers
Swap:   838444k total,  928500k used, 7455944k free, 57239612k cached

 PID USER      PR  NI  VIRT  RES  SHR  S %CPU %MEM    TIME+  COMMAND
19383 anakano  20   0   562m  18m 3328  R 100.4   0.1  1:35.33 hmd
19382 anakano  20   0   562m  18m 3528  R 100.1   0.1  1:35.24 hmd
19384 anakano  20   0   562m  18m 3328  R 100.1   0.1  1:35.32 hmd
19381 anakano  20   0   562m  18m 3552  R  99.7   0.1  1:35.27 hmd
19418 anakano  20   0  13400  1484  908  R   0.7   0.0  0:00.29 top
   1 root      20   0  19232   996  800  S   0.0   0.0  0:15.69 init
...
```
Instead of the interactive PBS job, you can 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
#PBS -l walltime=00:00:59
#PBS -o hmd.out
#PBS -j oe
#PBS -N hmd
WORK_HOME=/home/rcf-proj/anl/anakano
cd $WORK_HOME
cat $PBS_NODEFILE | uniq > nodefile
np=$(cat nodefile | wc -l)
mpirun -bind-to none -np $np -machinefile nodefile ./hmd
```

This way is recommended!
Effect of Core Binding

1. mpirun ./hmd --bind-to none -np 2 -machinefile nodefile

   [anakano@hpc2566 ~]$ top (then type H)
   top - 22:14:36 up 4 days, 12:17, 1 user, load average: 2.43, 2.38, 1.92
   Tasks: 260 total, 5 running, 255 sleeping, 0 stopped, 0 zombie
   Cpu(s): 49.9%us, 0.2%sy, 0.0%ni, 49.8%id, 0.0%wa, 0.0%hi, 0.1%si, 0.0%st
   Mem: 16331532k total, 1125796k used, 15205736k free, 173232k buffers
   Swap: 8388604k total, 0k used, 8388604k free, 566420k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
   18069 anakano  20   0 292m  18m 2944 R 100.0  0.1   0:12.48 hmd
   18073 anakano  20   0 292m  18m 2944 R 100.0  0.1   0:12.33 hmd
   18074 anakano  20   0 292m  18m 2944 R 100.0  0.1   0:12.38 hmd
   18075 anakano  20   0 292m  18m 2944 R 100.0  0.1   0:12.38 hmd
   24305 root      20   0 100m  52m  10m S  0.7  0.3 43:21.32 pbs_mom
   ...

2. mpirun ./hmd -np 2 -machinefile nodefile

   [anakano@hpc2566 ~]$ top (then type H)
   top - 22:27:38 up 4 days, 12:30, 1 user, load average: 2.04, 0.71, 1.01
   Tasks: 260 total, 2 running, 258 sleeping, 0 stopped, 0 zombie
   Cpu(s): 12.5%us, 0.2%sy, 0.0%ni, 87.3%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
   Mem: 16331532k total, 1125888k used, 15205644k free, 173396k buffers
   Swap: 8388604k total, 0k used, 8388604k free, 566428k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
   18407 anakano  20   0 292m  18m 2944 R 29.5  0.1   0:12.14 hmd
   18411 anakano  20   0 292m  18m 2944 S 23.6  0.1   0:09.62 hmd
   18412 anakano  20   0 292m  18m 2944 S 23.6  0.1   0:09.66 hmd
   18413 anakano  20   0 292m  18m 2944 S 23.6  0.1   0:09.74 hmd
   17787 root      20   0 280m 7336 2344 S  1.0  0.0 0:00.69 pvfs2-server
   ...
Strong Scalability of Hybrid MD

1 MPI process; 1-8 threads

In hmd.h:

\[ \text{vproc} = \{1,1,1\}, \text{nproc} = 1; \]
\[ \text{vthrd} = \{1,1,1\}, \text{nthrd} = 1; \]

\[
\begin{align*}
2 & 1 & 1 & 2 \\
2 & 2 & 1 & 4 \\
2 & 2 & 2 & 8
\end{align*}
\]

\[ \text{InitUcell[]} = \{24,24,24\} \]

\[ N = 4 \times 24^3 \]

\[ = 55296 \text{ atoms} \]

\[ P: \text{Number of cores} \]

\[ S_P = \frac{T(N,1)}{T(N,P)} \]

\[ E_P = \frac{S_P}{P} \]
Atomic Operation

- Restore Newton’s third law & handle race conditions with the omp atomic directive

```c
int bintra;
...
if (i<j && rr<rrCut) {
    ...
    if (bintra)
        lpe_td[std] += vVal;
    else
        lpe_td[std] += 0.5*vVal;
    for (a=0; a<3; a++) {
        f = fcVal*dr[a];
        ra[i][a] += f;
        if (bintra) {
            #pragma omp atomic
            ra[j][a] -= f; // Different threads can access the same atom
        }
    }
}
```
Atomic Operation Is Expensive

![Graph showing the relationship between the number of threads and wall clock time for atomic and original operations. The graph indicates that atomic operations are more expensive in terms of wall clock time compared to original operations, especially as the number of threads increases.](image-url)
Improved Strong Scalability of Hybrid MD

1 MPI process; 1-8 threads

In hmd.h:

\[ \text{vproc} = \{1,1,1\}, \text{nproc} = 1; \]
\[ \text{vthrd} = \{1,1,1\}, \text{nthrd} = 1; \]

\[
\begin{array}{c}
2 & 1 & 1 & 2 \\
2 & 2 & 1 & 4 \\
2 & 2 & 2 & 8 \\
\end{array}
\]

#PBS -l nodes=1:ppn=8

mpirun ... –np 1 ... ./hmd1

\[ P: \text{Number of cores} \]

InitUcell[] = \{24,24,24\}

\[ N = 4 \times 24^3 \]
\[ = 55296 \text{ atoms} \]
define shared;

... if used here

```c
#pragma omp parallel private(if used in both)
{
    define private;

    ... if only used (in left-hand side) here
}
```

... or here
Spatially Compact Thread Scheduling

Concurrency-control mechanism: Data privatization

- **Reduced memory:** $\Theta(nq) \rightarrow \Theta(n+n^{2/3}q^{1/3})$
- **Strong scaling parallel efficiency** 0.9 on quad quad-core AMD Opteron
- **2.6x speedup** over MPI by hybrid MPI+OpenMP on 32,768 IBM Blue Gene/P cores

M. Kunaseth et al., PDPTA’11; J. Supercomput. ('13)
Concurrency-Control Mechanisms

A number of concurrency-control mechanisms (CCMs) are provided by OpenMP to coordinate multiple threads:

- **Critical section**: Serialization
- **Atomic update**: Expensive hardware instruction
- **Data privatization**: Requires large memory $\Theta(nq)$
- **Hardware transactional memory**: Rollbacks (on IBM Blue Gene/Q)

CCM performance varies:

- **Depending on computational characteristics of each program**
- **In many cases, CCM degrades performance significantly**

---

**HTM/critical section**

```c
#pragma omp <critical|tm_atomic>
{
    ra[i][0] += fa*dr[0];
    ra[i][1] += fa*dr[1];
    ra[i][2] += fa*dr[2];
}
```

**Atomic update**

```c
#pragma omp atomic
    ra[i][0] += fa*dr[0];
#pragma omp atomic
    ra[i][1] += fa*dr[1];
#pragma omp atomic
    ra[i][2] += fa*dr[2];
```

**Data privatization**

Goal: Provide a guideline to choose the “right” CCM
Hardware Transactional Memory

Transactional memory (TM): An opportunistic CCM

- Avoids memory conflicts by monitoring a set of speculative operations (*i.e.* transaction)

- If two or more transactions write to the same memory address, transaction(s) will be restarted—_a process called rollback_

- If no conflict detected in the end of a transaction, operations within the transaction becomes permanent (*i.e.* committed)

- Software TM usually suffers from large overhead

Hardware TM on IBM Blue Gene/Q:

- The first commercial platform implementing TM support at hardware level via multiversioned L2-cache

- Hardware support is expected to reduce TM overhead

- Performance of HTM on molecular dynamics has not been quantified
Strong-Scaling Benchmark for MD

1 million particles on 64 Blue Gene/Q nodes with 16 cores per node

Developed a fundamental understanding of CCMs:

- **OMP-critical** has limited scalability on larger number of threads \((q > 8)\)
- Data privatization is the fastest, but it requires \(\Theta(nq)\) memory
- **Fused HTM** performs the best among constant-memory CCMs

M. Kunaseth et al., *PDSEC’13* Best Paper
Threading Guideline for Scientific Programs

Focus on minimizing runtime (best performance):

- Have enough memory → data privatization
- Conflict region is small → OMP-critical
- Small amount of updates → OMP-atomic
- Conflict rate is low → HTM
- Other → OMP-critical* (poor performance)

### Concurrency control mechanism

<table>
<thead>
<tr>
<th>Concurrency control mechanism</th>
<th>Parallel efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMP-critical</td>
<td>( e = \min(\frac{1}{p f_{CR}}, 1) )</td>
</tr>
<tr>
<td>OMP-atomic</td>
<td>( e = \frac{t_{total}}{t_{total} + m \mu c_{atomic}} )</td>
</tr>
<tr>
<td>Data privatization</td>
<td>( e = \frac{t_{total}}{t_{total} + c_{reduction} n \log p} )</td>
</tr>
<tr>
<td>HTM</td>
<td>( e = \frac{t_{total}}{t_{total} + m (c_{HTM_overhead} + \mu c_{HTM_update})} )</td>
</tr>
</tbody>
</table>

M. Kunaseth et al., PDSEC’13 Best Paper
IEEE PDSEC’13 Best Paper

IEEE PDSEC
Best Paper Award

Performance Characteristics of Hardware Transactional Memory for Molecular Dynamics Application on BlueGene/Q
Manaschai Kunaseth, Rajiv Kalia, Aiichiro Nakano, Priya Vashishta, David Richards, James Gosli

The 14th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC-13), held in Boston, Massachusetts, USA, May 24, 2013.

National Research Council of Thailand
Best Dissertation Award