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 Slurm script:

```
mpirun -np 2
```

In the code:

```
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`

$$\pi = \int_0^1 \frac{4}{1+x^2} \, dx \approx \Delta \sum_{i=0}^{N-1} \frac{4}{1+x_i^2}$$
#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 %d tid %d sum = %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
  
- **Slurm script**
  
  ```bash
  #!/bin/bash
  #SBATCH --nodes=2
  #SBATCH --ntasks-per-node=1
  #SBATCH --cpus-per-task=2
  #SBATCH --time=00:00:59
  #SBATCH --output=hpi.out
  #SBATCH -A lc_an2
  WORK_HOME=/home/rcf-proj/an2/anakano
  cd $WORK_HOME
  srun -n $SLURM_NNODES ./hpi
  ```

- **Output**

  ```text
  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)

Big picture = who does what: loop index $\rightarrow$ thread map
Linked-List Cell Block

Variables
• $\text{vthrd}[0|1|2]$ = # of OpenMP threads per MPI process in the x|y|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|y|z direction that each thread is assigned.

In main():
omp_set_num_threads($\text{nthrd}$);

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 */
    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 */
}
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];
```
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]))++
    {
    }
    ...
}
```
Avoiding Critical Sections (1)

- Remove the critical section

\[
\text{if (bintra) } \text{lpe }+ = \text{ vVal; else } \text{lpe }+ = 0.5*\text{vVal;}
\]

by defining an array, \text{lpe\_td[nthrd]}, where each array element stores the partial sum of the potential energy by a thread

Data privatization: \text{cf.omp\_pi.c & hpi.c}
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[][] for preventing race conditions
```c
#define shared;

... if used here

#pragma omp parallel private(if used in both)
{
    define private;
    ... if only used (in left-hand side) here
}

... or here
```
Running HMD at HPC

Submit a batch job using the following Slurm script.

```bash
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=4
#SBATCH --time=00:01:59
#SBATCH --output=hmd.out
#SBATCH -A lc_an2

WORK_HOME=/home/rcf-proj/an2/anakano
cd $WORK_HOME
	srun -n 2 ./hmd
```
1. Interactively submit a Slurm job & wait until you are allocated nodes. (Note that you will be automatically logged in to one of the allocated nodes.)

```
$ salloc --nodes=2 --ntasks-per-node=1 --cpus-per-task=4 -t 29
salloc: Granted job allocation 1784580
salloc: Waiting for resource configuration
salloc: Nodes hpc[0966,0969] are ready for job
```

Begin SLURM Prolog Fri Sep 28 11:03:56 2018
Job ID: 1784580
Username: anakano
Accountname: lc_an2
Name: sh
Partition: quick
Nodes: hpc[0966,0969]
TasksPerNode: 1(x2)
CPUSPerTask: 4
TMPDIR: /tmp/1784580.quick
SCRATCHDIR: /staging/scratch/1784580
Cluster: uschpc
HSDA Account: false
End SLURM Prolog

[anakano@hpc0966 anakano]$ You are logged in to one of the allocated nodes
Interactively Running HMD at HPC (2)

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

$ srun -n 2 ./hmd

3. 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 hpc0966
[anakano@hpc0969 ~]$ top (then type H)

KiB Mem: 24512700 total, 16214580 free, 712256 used, 7585864 buff/cache
KiB Swap: 8388604 total, 8290972 free, 97632 used, 22798280 avail Mem

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>9265</td>
<td>anakano</td>
<td>20</td>
<td>0</td>
<td>209016</td>
<td>20168</td>
<td>3208</td>
<td>R</td>
<td>99.9</td>
<td>0.1</td>
<td>1:18.03</td>
<td>hmd</td>
</tr>
<tr>
<td>9281</td>
<td>anakano</td>
<td>20</td>
<td>0</td>
<td>209016</td>
<td>20168</td>
<td>3208</td>
<td>R</td>
<td>87.8</td>
<td>0.1</td>
<td>1:08.47</td>
<td>hmd</td>
</tr>
<tr>
<td>9282</td>
<td>anakano</td>
<td>20</td>
<td>0</td>
<td>209016</td>
<td>20168</td>
<td>3208</td>
<td>R</td>
<td>87.8</td>
<td>0.1</td>
<td>1:08.47</td>
<td>hmd</td>
</tr>
<tr>
<td>9280</td>
<td>anakano</td>
<td>20</td>
<td>0</td>
<td>209016</td>
<td>20168</td>
<td>3208</td>
<td>R</td>
<td>87.5</td>
<td>0.1</td>
<td>1:08.47</td>
<td>hmd</td>
</tr>
<tr>
<td>9243</td>
<td>anakano</td>
<td>20</td>
<td>0</td>
<td>160336</td>
<td>2328</td>
<td>1552</td>
<td>R</td>
<td>1.0</td>
<td>0.0</td>
<td>0:00.86</td>
<td>top</td>
</tr>
<tr>
<td>622</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>43604</td>
<td>3408</td>
<td>2136</td>
<td>S</td>
<td>0.3</td>
<td>0.0</td>
<td>25:16.07</td>
<td>xfsaid/sda3</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:51.15</td>
<td>systemd</td>
</tr>
<tr>
<td>2</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:03.91</td>
<td>kthreadd</td>
</tr>
<tr>
<td>3</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:17.43</td>
<td>ksoftirqd/0</td>
</tr>
</tbody>
</table>

Type 1 to show core-usage summary
Validation of Hybrid MD

2 MPI process; 4 threads

In hmd.h:

\[ \text{vproc} = \{1,1,2\}, \ nproc = 2; \]
\[ \text{vthrd} = \{2,2,1\}, \ nthrd = 4; \]

Make sure that the total energy is the same as that calculated by pmd.c using the same input parameters, at least for \(\sim 5-6\) digits

<table>
<thead>
<tr>
<th>Time (sec)</th>
<th>Temperature</th>
<th>Potential energy</th>
<th>Total energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.050000</td>
<td>0.877345</td>
<td>-5.137153</td>
<td>-3.821136</td>
</tr>
<tr>
<td>0.100000</td>
<td>0.462056</td>
<td>-4.513097</td>
<td>-3.820013</td>
</tr>
<tr>
<td>0.150000</td>
<td>0.510836</td>
<td>-4.587287</td>
<td>-3.821033</td>
</tr>
<tr>
<td>0.200000</td>
<td>0.527457</td>
<td>-4.611958</td>
<td>-3.820772</td>
</tr>
<tr>
<td>0.250000</td>
<td>0.518668</td>
<td>-4.598798</td>
<td>-3.820796</td>
</tr>
<tr>
<td>0.300000</td>
<td>0.529023</td>
<td>-4.614343</td>
<td>-3.820808</td>
</tr>
<tr>
<td>0.350000</td>
<td>0.532890</td>
<td>-4.620133</td>
<td>-3.820798</td>
</tr>
<tr>
<td>0.400000</td>
<td>0.536070</td>
<td>-4.624899</td>
<td>-3.820794</td>
</tr>
<tr>
<td>0.450000</td>
<td>0.539725</td>
<td>-4.630387</td>
<td>-3.820799</td>
</tr>
<tr>
<td>0.500000</td>
<td>0.538481</td>
<td>-4.628514</td>
<td>-3.820792</td>
</tr>
</tbody>
</table>

See the lecture on “order-invariant real-number summation”
Strong Scalability of Hybrid MD

1 MPI process; 1-8 threads

In hmd.h:

\[
\begin{align*}
\text{vproc} &= \{1,1,1\}, \ nproc = 1; \\
vthrd &= \{1,1,1\}, \ nthrd = 1; \\
2 &\quad 1 &\quad 1 \\
2 &\quad 2 &\quad 1 \\
2 &\quad 2 &\quad 2
\end{align*}
\]

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

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

\[= 4 \times 24^3 = 55296\ \text{atoms}\]

\[\text{pmdd.in}\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitUcell[3]</td>
<td>24 24 24</td>
</tr>
<tr>
<td>Density</td>
<td>0.8</td>
</tr>
<tr>
<td>InitTemp</td>
<td>1.0</td>
</tr>
<tr>
<td>DeltaT</td>
<td>0.005</td>
</tr>
<tr>
<td>StepLimit</td>
<td>100</td>
</tr>
<tr>
<td>StepAvg</td>
<td>101</td>
</tr>
</tbody>
</table>

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

Graphs showing wall-clock time, speedup, and efficiency vs. number of threads for different processors.
Improved Strong Scalability of Hybrid MD

1 MPI process; 1-8 threads

In hmd.h:

\[
\begin{align*}
\text{vproc} &= \{1,1,1\}, \quad \text{nproc} = 1; \\
\text{vthrd} &= \{1,1,1\}, \quad \text{nthrd} = 1; \\
2 & 1 1 \\
2 & 2 1 \\
2 & 2 2
\end{align*}
\]

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

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

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

\[E_P = \frac{S_P}{P}\]
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 wall clock time and number of threads. The graph compares atomic and original operations, with the atomic operation showing a significant increase in wall clock time as the number of threads increases.](image)
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
- 2.6× 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

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

```c
#include <omp.h>

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

```c
#include <omp.h>

#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];
```
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></th>
<th>Parallel efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMP-critical</td>
<td>( e = \min\left(\frac{1}{p f_{CR}}, 1\right) )</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} \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 Best Paper & Beyond

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 Glosli

The 14th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC-13), held in Boston, Massachusetts, USA, May 24, 2013.
It All Started as a CSCI596 Final Project