Scalable MPI Completion

9\textsuperscript{TH} JLESC Workshop
04/15/2019
Thananon “Arm” Patinyasakdikul
MPI Completion - Design

MPI_Waitall, MPI_Waitsome, MPI_Waitany
MPI_Testall, MPI_Testsome, MPI_Testany

MPI_Isend(.., reqs[0]);
MPI_Isend(.., reqs[1]);
MPI_Isend(.., reqs[2]);
...
...
MPI_Isend(.., reqs[n-1]);

MPI_Waitall(n, &reqs, &statuses);
MPI Completion – Sync object

Previous OMPI implementation

Current OMPI implementation

- O(1) checking
- Multithreaded benefit*

For loop

Reqs[0]
Reqs[1]
Reqs[2]
Reqs[n-1]

Progress()

Sync

int count;

Reqs[0]
Reqs[1]
Reqs[2]
Reqs[n-1]

Progress()
MPI Completion – Bad case

MPI_Isend(.., reqs[0]);
MPI_Isend(.., reqs[1]);
MPI_Isend(.., reqs[2]);
...
MPI_Isend(.., reqs[n-1]);

MPI_Waitsome(n, &reqs, &statuses);
...
MPI_Waitsome(n, &reqs, &statuses);

DRAWBACK: If the request is not completed, we detach it from sync.
**MPI Completion – Worst case**

```c
MPI_Init_thread(MULTIPLE);
MPI_Isend(.., reqs[0]);
MPI_Isend(.., reqs[1]);
MPI_Isend(..,reqs[2]);
...
MPI_Isend(..,reqs[n-1]);

MPI_Waitsome(n, &reqs, &statuses);
...
MPI_Waitsome(n, &reqs, &statuses);
```

**MPI space**

- Reqs[0]
- Reqs[1]
- Reqs[2]
- Reqs[n-1]

**Sync**

```c
int count;
Progress()
```

**DRAWBACK:** If the request is not completed, we *detach* it from sync.

**MPI Completion – Worst case**

**MPI space**

- Reqs[0]
- Reqs[1]
- Reqs[2]
- Reqs[n-1]

**Sync**

```c
int count;
Progress()
```

**DRAWBACK:** If the request is not completed, we *detach* it from sync.
MPI Completion – Not for MPI_Test*
*two costly to create an object for single use.

MPI_Isend(.., reqs[0]);
MPI_Isend(.., reqs[1]);
MPI_Isend(..,reqs[2]);
..
...
MPI_Isend(..,reqs[n-1]);

MPI_Testsome(n, &reqs, &statuses);
...
MPI_Testsome(n, &reqs, &statuses);
- Give users control of the sync object.
  - Reusable (No need for array rebuilding)
  - Relieves MPI from detaching.
  - Support user-level callback function*.

```c
int count;

MPIX_Sync

User space

MPI space

Reqs[0]
Reqs[1]
Reqs[2]
Reqs[n-1]

Sync
Progress()
MPIX_Sync- API

Initialize the object.

Attach the request with sync object and completion data.

Return the first completion_data associated with sync object.

Return n completion_data associated with sync object. (testsome API)
MPIX.Sync API

int MPIX_Sync_waitall(MPIX_Sync sync);
*not sure if needed.

int MPIX_Sync_testall(MPIX_Sync sync);
*not sure if needed.

int MPIX_Sync_size(MPIX_Sync sync);

int MPIX_Sync_probe(MPIX_Sync sync);

int MPIX_Sync_free(MPIX Sync *sync);

Wait for all attached requests to complete.

Check if all attached requests are completed.

Get the number of incomplete requests.

Get the number of *unconsumed* completed requests.
MPIX_Sync_query – speedup

- Faster turnaround time if nothing completed $\sim O(1)$.
- Slightly slower if every request completed. (Tuning possible)
- MPIX_Sync_query_bulk gives slightly better performance.
- API provides more flexibilities.
MPIX_Sync_query – Application

*pelim results from PaRSEC (Runtime task scheduler)
- ~10-30 persistent requests + MPI_Testsome()

<table>
<thead>
<tr>
<th>message size (bytes)</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>13.73 %</td>
</tr>
<tr>
<td>4000</td>
<td>7.15 %</td>
</tr>
<tr>
<td>40000</td>
<td>3.1 %</td>
</tr>
<tr>
<td>400000</td>
<td>2.55 %</td>
</tr>
</tbody>
</table>

Pingpong (communication only)

<table>
<thead>
<tr>
<th>kernel</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpotrf</td>
<td>~0%*</td>
</tr>
<tr>
<td>dgeqrf</td>
<td>~0%*</td>
</tr>
</tbody>
</table>

DPLASMA (matrix factorization)

*within error margin
MPIX_Sync_query – User level callback

Discussion:

- Users can already call the function after query. (Passing a structure as completion_data)
  *more control to the user

- MPI can call callback function as soon as ..
  - A request is completed
  - Certain number of requests are completed.
  *might affect communication performance

```c
int MPIX_Sync_init(MPIX_Sync *sync);

int MPIX_Sync_attach(MPIX_Sync sync,
                     MPI_Request request,
                     void *completion_data);

void* MPIX_Sync_query(MPIX_Sync sync,
                       MPI_Status *status);

int MPIX_Sync_query_bulk(int incount;
                          MPIX_Sync sync,
                          int *outcount;
                          void **completion_data
                          MPI_Status *status);
```
MPIX_Sync- API

```c
typedef void MPIX_Sync;

int MPIX_Sync_init(MPIX_Sync *sync);
int MPIX_Sync_attach(
    MPIX_Sync sync,
    MPI_Request request,
    void *completion_data);

void* MPIX_Sync_query(
    MPIX_Sync sync,
    MPI_Status *status);

int MPIX_Sync_query_bulk(
    int incount;
    MPIX_Sync sync,
    int *outcount;
    void **completion_data
    MPI_Status *status);
```

- `completion_data` can be MPIX_SYNC_NO_COMPLETION_DATA.
- Return MPIX_SYNC_EMPTY if there is no completion.
- `incount` can be any number. (Unlike MPI_Testsome where it has to be nrequests)
- Return MPIX_SYNC_MORE if there is more completion than asked for.
MPIX_Sync - API

- User relinquish the request in attach.
- We free the request here.

Users are responsible to keep track of the requests before passing it to attach if they want to cancel them.