universidade de aveiro

Computação Paralela
Módulo MPI
2021/2022

Rui Costa

Email: americo.costa@ua.pt

Manager-worker prototype

universidade de aveiro

- One of the processes, called *manager*, is responsible for coordinating the work of the other processes, called *workers*.
- This kind of algorithm is especially appropriate when:
  - *worker* processes do not have to communicate with each other,
  - and the amount of work to be performed by each *worker* is difficult to predict.
- Communications will be made individually between the manager and each of the workers (point-to-point communications).

A self-scheduling example: Matrix-vector multiplication

universidade de aveiro

- Given a matrix $\hat{A}$ and a vector $\vec{b}$, calculate the vector $\vec{c}$ resulting from the product of $\hat{A}$ by $\vec{b}$:

$$\vec{c} = \hat{A}\vec{b}.$$

- The unit of work to be given out by the *manager* to the *workers* consists of the dot product between a row of matrix $\hat{A}$ by the vector $\vec{b}$, which returns

$$c_i = \sum_j A_{ij} b_i.$$

Self-scheduling matrix-vector multiplication algorithm

universidade de aveiro

Manager Part

- The *manager* begins by broadcasting $\vec{b}$ to all *workers*.
- Initially the *manager* sends a row of $\hat{A}$ to each *worker*, and then starts a loop which will terminate when all of the c_i's have been received.
- In each step of the loop the *manager* receives a c_i from whichever *worker* sends one first, and sends the next task (row of $\hat{A}$) to that *worker*.
- Once all tasks have been handed out to the *workers,* termination messages are sent instead.

Self-scheduling matrix-vector multiplication algorithm

universidade de aveiro

Worker Part

- After each *worker* receives the broadcast of vector $\vec{b}$, it also enters a loop.
- In each step of the loop the *worker*
  i. receives a row of $\hat{A}$,
  ii. calculates the dot product of that row with $\vec{b}$,
  iii. and sends the result back to the *manager*.
- The *worker* exits the loop when the termination message is received from the *manager*.

Self-scheduling matrix-vector multiplication algorithm

universidade de aveiro

The code for this program is divided in three parts: the *manager* and *worker* parts described above, and the part that is common to both *manager* and *workers.*

Common part

- MPI initialization
- Variable declarations and initializations
- Memory allocations
- MPI_Finalize()

Self-scheduling sends and receives

universidade de aveiro

The distinctive feature of self-scheduling programs is that the *manager* is prepared to receive messages from whichever *worker* sends one first.

So, the receive function called by the *manager* must allow for the message to arrive from any *worker* (`source`) with any `tag`:

```
MPI_Recv(&ans, 1, MPI_INT, MPI_ANY_SOURCE,
MPI_ANY_TAG, MPI_COMM_WORLD, &status);
```

Nevertheless, the *manager* still needs to know who was the `source` of the message (which is also destination of the next task), and the `tag` with which the message was sent (the `tag` is used to tell the *manager* where to store `ans`, ie. `tag` is the matrix row's index).

Self-scheduling sends and receives

Similarly, the receive function of the *worker* must allow for any `tag` (matrix row) of the received message:

```
MPI_Recv(row, ncols, MPI_INT, 0, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
```

(Here the rank of the *manager* is `0`.)

The actual `source` and `tag` of a message received can be retrieved from the `status` parameter as:

```
source = status.MPI_SOURCE
tag = status.MPI_TAG
```

Self-scheduling sends and receives

Beware, the sends must always indicate the destination and `tag` of the message, both for the *manager*

```
MPI_Send(row, ncols, MPI_INT, worker_rank,
row_index, MPI_COMM_WORLD);
```

and for the *workers*

```
MPI_Send(&ans, 1, MPI_INT, 0, row_index,
MPI_COMM_WORLD);
```

Additional notes

- Define/load matrix $\hat{A}$ and vector $\vec{b}$ in the *manager* only.
- Use collective `MPI_Bcast` to pass $\vec{b}$ onto the *workers*:

```
MPI_Bcast(b, ncols, MPI_INT, manager_rank,
MPI_COMM_WORLD);
```

- Termination messages to the *workers* will be sent with a particular value of `tag` different from all possible values of `row_index`, for example `tag_term = nrows+1`:

```
MPI_Send(MPI_BOTTOM, 0, MPI_INT, worker_rank,
tag_term, MPI_COMM_WORLD);
```

- Allow for a number of rows smaller than the number of *workers*.

Bibliography

universidade de aveiro

Using MPI: portable parallel programming with the message-passing interface, 3rd edition, William Gropp, Ewing Lusk, and Anthony Skjellum, MIT press (2014).