spmd

Execute code in parallel on workers of parallel pool

Syntax

```
spmd
    statements
end
```

Description

`spmd, statements, end` defines an `spmd` statement on a single line. MATLAB® executes the `spmd` body denoted by `statements` on several MATLAB workers simultaneously. Each worker can operate on a different data set or different portion of distributed data, and can communicate with other participating workers while performing the parallel computations. The `spmd` statement can be used only if you have Parallel Computing Toolbox™. To execute the statements in parallel, you must first create a pool of MATLAB workers using `parpool` or have your parallel preferences allow the automatic start of a pool. example

Inside the body of the `spmd` statement, each MATLAB worker has a unique value of `labindex`, while `numlabs` denotes the total number of workers executing the block in parallel. Within the body of the `spmd` statement, communication functions for communicating jobs (such as `labSend` and `labReceive`) can transfer data between the workers.

Values returning from the body of an `spmd` statement are converted to `Composite` objects on the MATLAB client. A Composite object contains references to the values stored on the remote MATLAB workers, and those values can be retrieved using cell-array indexing. The actual data on the workers remains available on the workers for subsequent `spmd` execution, so long as the Composite exists on the client and the parallel pool remains open.

By default, MATLAB uses all workers in the pool. When there is no pool active, MATLAB will create a pool and use all the workers from that pool. If your preferences do not allow automatic pool creation, MATLAB executes the block body locally and creates Composite objects as necessary. You cannot execute an `spmd` block if any worker is busy executing a `parfeval` request, unless you use `spmd(0)`.

For more information about `spmd` and Composite objects, see Distribute Arrays and Run SPMD.

Note

Use `parfevalOnAll` instead of `parfor` or `spmd` if you want to use `clear`. This preserves workspace transparency. See Ensure Transparency in parfor-Loops or spmd Statements.

`spmd(n), statements, end` uses `n` to specify the exact number of MATLAB workers to evaluate `statements`, provided that `n` workers are available from the parallel pool. If there are not enough workers available, an error is thrown. If `n` is zero, MATLAB executes the block body locally and creates Composite objects, the same as if there is no pool available. example

`spmd(m,n), statements, end` uses a minimum of `m` and a maximum of `n` workers to evaluate `statements`. If there are not enough workers available, an error is thrown. `m` can be zero, which allows the block to run locally if no workers are available. example

Examples

collapse all

Execute Code in Parallel with `spmd`

Create a parallel pool, and perform a simple calculation in parallel using `spmd`. MATLAB executes the code inside the `spmd` on all workers in the parallel pool.

View MATLAB Command

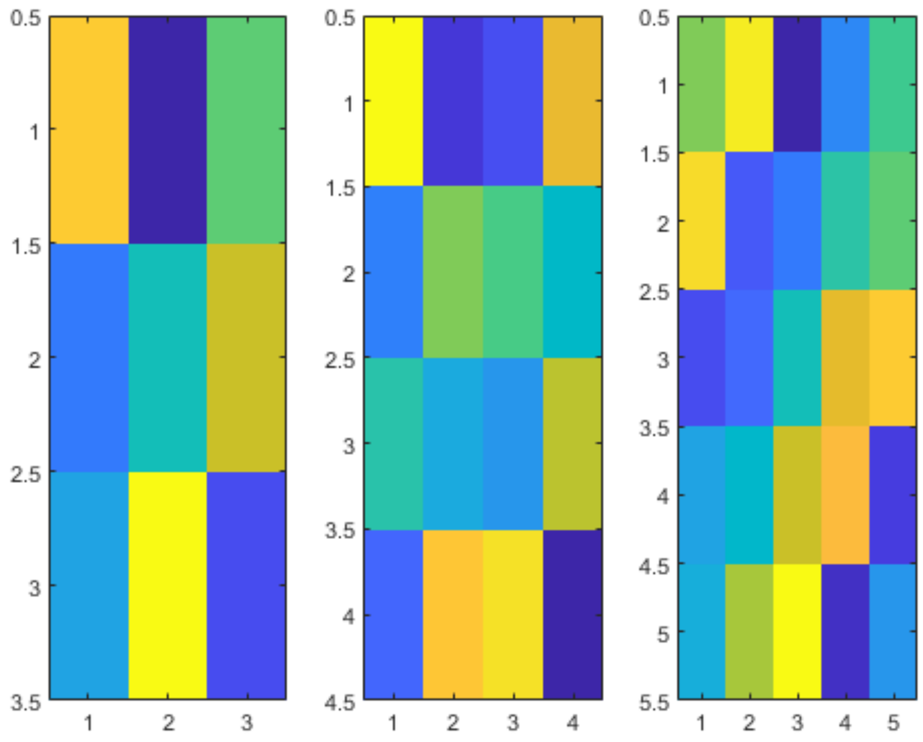
```
parpool(3);
```

```
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 3).
```

```
spmd
  q = magic(labindex + 2);
end
```

Plot the results.

```
figure
subplot(1,3,1), imagesc(q{1});
subplot(1,3,2), imagesc(q{2});
subplot(1,3,3), imagesc(q{3});
```



When you are done with computations, you can delete the current parallel pool.

```
delete(gcp);
```

Use Multiple GPUs in a Parallel Pool

If you have access to several GPUs, you can perform your calculations on multiple GPUs in parallel using a parallel pool.

Start a parallel pool with as many workers as GPUs. To determine the number of GPUs available, use the gpuDeviceCount function. By default, MATLAB assigns a different GPU to each worker for best performance.

```
parpool('local',gpuDeviceCount);
```

To identify which GPU each worker is using, call gpuDevice inside an spmd block. The spmd block runs gpuDevice on every worker.

```
spmd
    gpuDevice
end
```

Use parallel language features, such as parfor or parfeval, to distribute your computations to workers in the parallel pool. If you use gpuArray enabled functions in your computations, these functions run on the GPU of the worker. For more information, see Run MATLAB Functions on a GPU. For an example, see Run MATLAB Functions on Multiple GPUs.

When you are done with your computations, shut down the parallel pool. You can use the gcp function to obtain the current parallel pool.

```
delete(gcp('nocreate'));
```

If you want to use a different choice of GPUs, you can use gpuDevice to select a particular GPU on each worker. Define an array, for example gpuIndices, that contains the indices of the GPUs to activate on each worker. Then, start a parallel pool with as many workers as GPUs to select, and use an spmd block to run gpuDevice on each worker. The labindex function identifies each worker. Use this function to associate a worker with a GPU index.

```
gpuIndices = [1 3];
parpool(numel(gpuIndices));
spmd
    gpuDevice(gpuIndices(labindex));
end
```

As a best practice, and for best performance, assign a different GPU to each worker.

Tips

- An spmd block runs on the workers of the existing parallel pool. If no pool exists, spmd will start a new parallel pool, unless the automatic starting of pools is disabled in your parallel preferences. If there is no parallel pool and spmd cannot start one, the code runs serially in the client session.
- If the AutoAttachFiles property in the cluster profile for the parallel pool is set to true, MATLAB performs an analysis on an spmd block to determine what code files are necessary for its execution, then automatically attaches those files to the parallel pool job so that the code is available to the workers.
- For information about restrictions and limitations when using spmd, see Run Single Programs on Multiple Data Sets.

See Also

Composite | batch | gop | labindex | numlabs | parallel.pool.Constant | parpool

Introduced in R2008b