parpool

Create parallel pool on cluster

Syntax

```
parpool
parpool(poolsize)
parpool(resources)
parpool(resources,poolsize)
parpool( __ ,Name,Value)
poolobj = parpool( __ )
```

Description

`parpool` starts a parallel pool of workers using the default cluster profile. With default preferences, MATLAB® starts a pool on the local machine with one worker per physical CPU core, up to the preferred number of workers. For more information on parallel preferences, see Specify Your Parallel Preferences. example

In general, the pool size is specified by your parallel preferences and the default profile. `parpool` creates a pool on the default cluster with its `NumWorkers` in the range `[1, preferredNumWorkers]` for running parallel language features. `preferredNumWorkers` is the value defined in your parallel preferences. For all factors that can affect your pool size, see Pool Size and Cluster Selection.

`parpool` enables the full functionality of the parallel language features in MATLAB by creating a special job on a pool of workers, and connecting the MATLAB client to the parallel pool. Parallel language features include `parfor`, `parfeval`, `parfevalOnAll`, `spmd`, and `distributed`. If possible, the working folder on the workers is set to match that of the MATLAB client session.

`parpool(poolsize)` creates and returns a pool with the specified number of workers. `poolsize` can be a positive integer or a range specified as a 2-element vector of integers. If `poolsize` is a range, the resulting pool has size as large as possible in the range requested. example

Specifying the `poolsize` overrides the number of workers specified in the preferences or profile, and starts a pool of exactly that number of workers, even if it has to wait for them to be available. Most clusters have a maximum number of workers they can start. If the profile specifies a MATLAB Job Scheduler cluster, `parpool` reserves its workers from among those already running and available under that MATLAB Job Scheduler. If the profile specifies a local or third-party scheduler, `parpool` instructs the scheduler to start the workers for the pool.

`parpool(resources)` or `parpool(resources,poolsize)` starts a worker pool on the resources specified by `resources`. example

`parpool( __ ,Name,Value)` applies the specified values for certain properties when starting the pool. example

`poolobj = parpool( __ )` returns a parallel.Pool object to the client workspace representing the pool on the cluster. You can use the pool object to programmatically delete the pool or to access its properties. Use `delete(pool)` to shut down the parallel pool. example

Examples

collapse all

Create Pool from Default Profile

Start a parallel pool using the default profile to define the number of workers. With default preferences, the

default pool is on the local machine.

```
parpool
```

Create Pool on Local Machine

You can create pools on different types of parallel environments on your local machine.

- Start a parallel pool of process workers.

  ```
  parpool('local')
  ```

- Start a parallel pool of thread workers.

  ```
  parpool('threads')
  ```

Fore more information on parallel environments, see Choose Between Thread-Based and Process-Based Environments.

Create Pool from Specified Profile

Start a parallel pool of 16 workers using a profile called `myProf`.

```
parpool('myProf',16)
```

Create Pool on Specified Cluster

Create an object representing the cluster identified by the default profile, and use that cluster object to start a parallel pool. The pool size is determined by the default profile.

```
c = parcluster
parpool(c)
```

Create Pool and Attach Files

Start a parallel pool with the default profile, and pass two code files to the workers.

```
parpool('AttachedFiles',{'mod1.m','mod2.m'})
```

Return Pool Object and Delete Pool

Create a parallel pool with the default profile, and later delete the pool.

```
poolobj = parpool;

delete(poolobj)
```

Determine Size of Current Pool

Find the number of workers in the current parallel pool.

```
poolobj = gcp('nocreate'); % If no pool, do not create new one.
if isempty(poolobj)
    poolsize = 0;
else
    poolsize = poolobj.NumWorkers
end
```

Input Arguments

collapse all

`poolsize` — Size of parallel pool

positive integer | 2-element vector of integers

Size of the parallel pool, specified as a positive integer or a range specified as a 2-element vector of integers. If `poolsize` is a range, the resulting pool has size as large as possible in the range requested. Set the default preferred number of workers in the parallel preferences or parallel profile.

Example: `parpool('local',2)`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`resources` — Resources to start pool on

`'local'` (default) | `'threads'` | profile name | cluster object

Resources to start the pool on, specified as `'local'`, `'threads'`, a cluster profile name or cluster object.

- `'local'` – Starts a pool of process workers on the local machine. For more information on process-based environments, see Choose Between Thread-Based and Process-Based Environments.
- `'threads'` – Starts a pool of thread workers on the local machine. For more information on thread-based environments, see Choose Between Thread-Based and Process-Based Environments.
- Profile name – Starts a pool on the cluster specified by the profile. For more information on cluster profiles, see Discover Clusters and Use Cluster Profiles.
- Cluster object – Starts a pool on the cluster specified by the cluster object. Use `parcluster` to get a cluster object.

Example: `parpool('local')`

Example: `parpool('threads')`

Example: `parpool('myClusterProfile',16)`

Example: `c = parcluster; parpool(c)`

Data Types: `char` | `string` | `parallel.Cluster`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'AttachedFiles',{'myFun.m'}`

`'AttachedFiles'` — Files to attach to pool

character vector | string scalar | string array | cell array of character vectors

Files to attach to pool, specified as a character vector, string or string array, or cell array of character vectors.

With this argument pair, `parpool` starts a parallel pool and passes the identified files to the workers in the pool. The files specified here are appended to the `AttachedFiles` property specified in the applicable parallel profile to form the complete list of attached files. The `'AttachedFiles'` property name is case sensitive, and must appear as shown.

Example: `{'myFun.m','myFun2.m'}`

Data Types: `char` | `cell`

`'AutoAddClientPath'` — Specifies if client path is added to worker path

true (default) | false

A logical value (`true` or `false`) that controls whether user-added-entries on the client's path are added to each worker's path at startup. By default 'AutoAddClientPath' is set to `true`.

Data Types: `logical`

`'EnvironmentVariables'` — Environment variables copied to workers

character vector | string scalar | string array | cell array of character vectors

Names of environment variables to copy from the client session to the workers, specified as a character vector, string or string array, or cell array of character vectors. The names specified here are appended to the `'EnvironmentVariables'` property specified in the applicable parallel profile to form the complete list of environment variables. Any variables listed which are not set are not copied to the workers. These environment variables are set on the workers for the duration of the parallel pool.

Data Types: `char` | `cell`

`'SpmdEnabled'` — Indication if pool is enabled to support SPMD
true (default) | false

Indication if pool is enabled to support SPMD, specified as a logical. You can disable support only on a local or MATLAB Job Scheduler cluster. Because `parfor` iterations do not involve interworker communication, disabling SPMD support this way allows the parallel pool to keep evaluating a `parfor`-loop even if one or more workers aborts during loop execution.

Data Types: `logical`

`'IdleTimeout'` — Time after which the pool shuts down if idle
nonnegative integer

Time in minutes after which the pool shuts down if idle, specified as an integer greater than zero. A pool is idle if it is not running code on the workers. By default `'IdleTimeout'` is the same as the value in your parallel preferences. For more information on parallel preferences, see Specify Your Parallel Preferences.

Example: `pool = parpool('IdleTimeout',120)`

Output Arguments

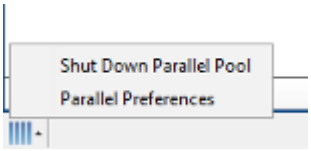
collapse all

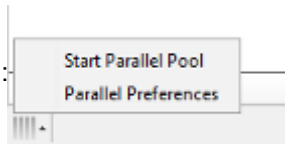
`poolobj` — Access to parallel pool from client
parallel.Pool object

Access to parallel pool from client, returned as a parallel.Pool object.

Tips

- The pool status indicator in the lower-left corner of the desktop shows the client session connection to the pool and the pool status. Click the icon for a menu of supported pool actions.

  With a pool running: 

  With no pool running: 

- If you set your parallel preferences to automatically create a parallel pool when necessary, you do not need to explicitly call the `parpool` command. You might explicitly create a pool to control when you incur the overhead time of setting it up, so the pool is ready for subsequent parallel language constructs.
- `delete(poolobj)` shuts down the parallel pool. Without a parallel pool, `spmd` and `parfor` run as a single

thread in the client, unless your parallel preferences are set to automatically start a parallel pool for them.

- When you use the MATLAB editor to update files on the client that are attached to a parallel pool, those updates automatically propagate to the workers in the pool. (This automatic updating does not apply to Simulink® model files. To propagate updated model files to the workers, use the `updateAttachedFiles` function.)
- If possible, the working folder on the workers is initially set to match that of the MATLAB client session. Subsequently, the following commands entered in the client Command Window also execute on all the workers in the pool:
  - `cd`
  - `addpath`
  - `rmpath`

  This behavior allows you to set the working folder and the command search path on all the workers, so that subsequent pool activities such as `parfor`-loops execute in the proper context.

  When changing folders or adding a path with `cd` or `addpath` on clients with Windows® operating systems, the value sent to the workers is the UNC path for the folder if possible. For clients with Linux® operating systems, it is the absolute folder location.

  If any of these commands does not work on the client, it is not executed on the workers either. For example, if `addpath` specifies a folder that the client cannot access, the `addpath` command is not executed on the workers. However, if the working folder can be set on the client, but cannot be set as specified on any of the workers, you do not get an error message returned to the client Command Window.

  Be careful of this slight difference in behavior in a mixed-platform environment where the client is not the same platform as the workers, where folders local to or mapped from the client are not available in the same way to the workers, or where folders are in a nonshared file system. For example, if you have a MATLAB client running on a Microsoft® Windows operating system while the MATLAB workers are all running on Linux operating systems, the same argument to `addpath` cannot work on both. In this situation, you can use the function `pctRunOnAll` to assure that a command runs on all the workers.

  Another difference between client and workers is that any `addpath` arguments that are part of the `matlabroot` folder are not set on the workers. The assumption is that the MATLAB install base is already included in the workers' paths. The rules for `addpath` regarding workers in the pool are:
  - Subfolders of the `matlabroot` folder are not sent to the workers.
  - Any folders that appear before the first occurrence of a `matlabroot` folder are added to the top of the path on the workers.
  - Any folders that appear after the first occurrence of a `matlabroot` folder are added after the `matlabroot` group of folders on the workers' paths.

  For example, suppose that `matlabroot` on the client is `C:\Applications\matlab\`. With an open parallel pool, execute the following to set the path on the client and all workers:

```
addpath('P1',
        'P2',
        'C:\Applications\matlab\T3',
        'C:\Applications\matlab\T4',
        'P5',
        'C:\Applications\matlab\T6',
        'P7',
        'P8');
```

  Because `T3`, `T4`, and `T6` are subfolders of `matlabroot`, they are not set on the workers' paths. So on the workers, the pertinent part of the path resulting from this command is:

```
P1
P2
<worker original matlabroot folders...>
P5
P7
P8
```

- If you are using Macintosh or Linux, and see problems during large parallel pool creation, see Recommended System Limits for Macintosh and Linux.

See Also

`Composite` | `delete` | `distributed` | `gcp` | `parallel.defaultClusterProfile` | `parallel.pool.Constant` | `parcluster` | `parfeval` | `parfevalOnAll` | `parfor` | `pctRunOnAll` | `spmd`

Topics

Specify Your Parallel Preferences

Discover Clusters and Use Cluster Profiles

Pass Data to and from Worker Sessions

Introduced in R2013b