多核计算平台中 MATLAB 并行计算工具包

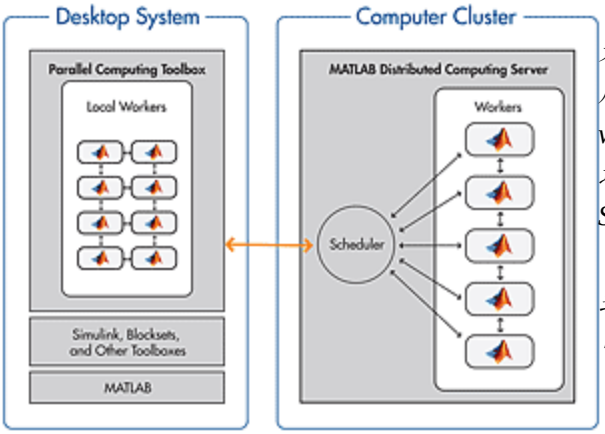
1 MATLAB 并行计算工具包简介

工程师和科学家们面临着用更少的时间建立复杂系统模型的需求，他们使用分布式和并行计算来解决高性能计算的问题。这些分布式的环境由多处理器和多核计算机来实现。

Mathworks 公司开发的分布式计算工具箱可以在多处理器计算环境中使用 MATLAB 和 Simulink 解决计算、数据密集型问题。可以使用该工具箱解决通过装配多个处理器包含几个单独工作单位或单个大型计算的问题。这些处理器可以驻留在一个多处理器计算机上，或者，当工具箱配合 MATLAB 分布式计算引擎时，驻留在计算机集群上。

该工具箱提供高级构造，如并行回路、并行算法、基于 MPI 的函数，以及用于作业和任务管理的低级构造。并行命令窗口提供熟悉的用于开发并行应用程序的 MATLAB 交互式环境。也能够在批处理环境中脱机执行分布式和并行应用程序。

利用 Parallel Computing Toolbox（并行计算工具箱），可在多核和多处理器计算机上使用 MATLAB 和 Simulink 来解决计算问题和数据密集型问题。并行处理结构包括并行 for 循环和代码块、分布式数组、并行数值算法，以及消息传递函数等，可以以较高的级别在 MATLAB 中执行任务及数据并行算法，而无需为特定的硬件和网络架构编写程序。这样，将串行 MATLAB 应用程序转换为 并行 MATLAB 应用程序，便几乎不需要修改代码，且不需要使用低级语言编写程序。此外，还可以在各种批处理环境中交互运行或脱机运行应用程序。



使用 Parallel Computing Toolbox 开发并行应用程序。利用该工具箱，应用程序能够在包含多达四个本地 worker（左）的桌面建立原型，并且，通过 MATLAB Distributed Computing Server（MATLAB 分布式计算服务器）（右），可以扩展应用程序，将其应用到一个集群上的多台计算机。单击图像可查看大图。

使用工具箱在单个多核或多处理器桌面上执行应用程序。无需更改代码，即可在计算机集群上运行同一个应用程序（使用 MATLAB Distributed Computing Serverâ„¢）。并行的 MATLAB 应用程序可以作为可执行程序或共享库（用 MATLAB Compilerâ„¢ 构建）分发，这些可执行程序或共享库可以访问 MATLAB Distributed Computing Server。

主要功能

- 支持数据并行和任务并行的应用程序开发

- 可使用 parfor（并行 for 循环）和 spmd（单程序多数据）注释代码段，用于执行数据并行和任务并行的算法
- 高级别的结构，如分布式数组、并行算法，以及消息传递函数，可在多个处理器上处理大型的数据集
- 可在一个多核桌面上本地运行八个 worker
- 与 MATLAB Distributed Computing Server 集成，可用于使用调度程序或任意数量 worker 的基于集群的应用程序
- 提供交互模式和批量执行模式

Parallel Computing Toolbox（并行计算工具箱）提供了多种高级编程结构，利用此工具箱，可对串行 MATLAB 代码进行转换，使之在数个 *worker*（独立于 MATLAB 客户端运行的 MATLAB 计算引擎）上并行运行。这些 worker 既可在桌面上运行（工具箱在桌面上可本地运行多达八个 worker），也可在集群上运行（使用 MATLAB Distributed Computing Server（MATLAB 分布式计算服务器））。这样的结构可降低在 MATLAB 客户端与 worker 之间、以及各 worker 之间管理计算与数据的协调与分发的复杂性，从而简化并行代码开发。此外，可使用关键词，例如 parfor（并行 for 循环）和 spmd（单程序多数据）语句来注释 MATLAB 代码，从而可以探究算法的各个区段所提供的任务和数据并行机制。

MATLAB 池和并行命令窗口 (pmode) 支持交互执行，这样，在设置耗时较长的运行或脱机任务前，可以先测试应用程序的某些区段。即使没有 worker，这些结构仍能够运作，这样，只需维护单个代码版本，即可确保串行执行和并行执行。

通过将 Monte Carlo 仿真和其他粗粒度或密集并行问题组织为独立的任务（工作单元），即可实现其并行化。工具箱中的并行 for 循环提供了一种在多个 MATLAB worker 间分配任务的方式。使用该循环，可以将独立的循环迭代自动分配给多个 MATLAB worker。parfor 结构管理着 MATLAB 客户端会话与 worker 之间的数据和代码传输。它会自动检测是否有 worker，如果没有，则会还原为串行方式。

此外，还可以将任务编写为 MATLAB 函数或 MATLAB 脚本。如果指定为函数，可以通过在工具箱中处理任务和作业对象来执行任务；如果指定为脚本，则可以使用批处理功能。

对于需要大型数据集处理的 MATLAB 算法，Parallel Computing Toolbox 提供了分布式数组、并行函数，以及使用 spmd 关键词注释代码区段的功能，可用于在数个 worker 上并行执行。这些并行结构可处理 worker 间通信，并协调后台的并行计算。

使用分布式数组，可以在参与并行计算的所有 worker 间分配任何数据类型的矩阵。利用并行函数，可以执行多种数学运算，例如索引、矩阵相乘、分解，以及在分布式数组上直接转换。此外，工具箱还提供了 150 个多个用于分布式数组的函数，包括基于 ScaLAPACK 的线性代数例程。

如要对并行方案进行显式的、细粒度的控制，同时还需要明确管理 worker 间的协调，通过 Parallel Computing Toolbox 函数可访问基于 MPI 标准 (MPICH2) 的消息传递例程，包括用于发送、接收、广播、阻挡、以及探测操作的函数。

使用 spmd 结构，可以指定代码的区段以在所有参与并行计算的 worker 间并行运行。程序执行过程中，该结构会自动将在其内部使用的数据和代码传输给 worker，并在执行完毕后将结果返回给 MATLAB 客户端会话。分布式数组、并行函数以及消息传递函数可以在 spmd 结构内部使用。在没有 worker 的情况下，MATLAB 会串行执行 spmd 语句。

2 数据并行编程（parfor）

2.1 简介

MATLAB 软件中 parfor 循环(parfor-loop)的基本概念和标准 MATLAB for 循环(for-loop)是一样的: MATLAB 在一些值的范围内执行一系列语句(循环)。parfor 循环体一部分运行在 MATLAB client(parfor 分发的地方)，部分并行地运行在 MATLAB workers——大部分的计算工作在这里完成，得到的结果被发回 client，然后整合(发回的计算结果是零散的，在 client 整合为最终的结果)。

由于若干 MATLAB workers 可以在同一循环上同时计算，parfor 循环能够提供比与其类似的 for 循环好得多的性能。

parfor 循环体的每次执行都是一次迭代。MATLAB workers 以非特定顺序并相互独立地计算(evaluate)迭代。因为每次迭代都是独立的，无论如何也不能保证这些迭代的同步，也没有必要。如果 workers 的数量与迭代次数相等，那么每个 worker 执行一次迭代。如果迭代次数多于 workers 数量，那有些 workers 执行多次迭代；在这种情况下，为减少通信时间，一个 workers 可能一次接到多次迭代。

2.1.1 parfor 的使用场景

当你需要简单计算的多次循环迭代时，例如蒙特卡洛(Monte Carlo)模拟，parfor 循环就很有用。parfor 将循环迭代分组，那么每个 worker 执行迭代的一部分。当迭代耗时很长的时候 parfor 循环也是有用的，因为 workers 可以同时执行迭代。

当循环中有迭代依赖其他迭代的结果时不应该使用 parfor 循环。每个迭代都必须不依赖其他迭代。由于 parfor 循环内有通信消耗，当只有小数量的简单计算时使用 parfor 可能得不到什么好处。这部分的例子只是用来说明 parfor 循环的表现(行为)，不一定证明这些应用最适合它们(parfor 循环)。

2.1.2 使用 matlabpool 设置 MATLAB 资源

为执行随后的 parfor 循环，使用 matlabpool 函数预设 MATLAB workers 的数量。Workers 可以远程运行于集群上，或者就运行在本地 MATLAB client，这取决于你的安排。You identify a scheduler and cluster by selecting a parallel configuration.关于怎样管理和使用配置，参见 `Programming with User Configurations.`

开始这部分的例子，为评估循环迭代分配本地 MATLAB workers：

```
matlabpool
```

这个命令启动了由默认并行配置定义数量的 MATLAB workers。如果默认的本地配置，并且

没有指定 workers 的数量，则每个核心启动一个 worker。

NOTE 如果 matlabpool 没有运行，parfor 循环在 client 上串行运行，不考虑迭代顺序。

2.1.3 创建 parfor-Loop

关于 parfor 循环，最安全的假设就是循环的每次迭代都由不同的 MATLAB worker 计算。如果你有一个所有迭代都完全独立于其他迭代的 for 循环，那它很适合改为 parfor。基本上，如果一个迭代依赖另一个迭代的结果，这些迭代不是独立的，不能被并行计算，这个循环就不容易被转换为 parfor 循环。

下面的例子生成了相同的结果，左边的是 for 循环，右边的是 parfor 循环。每个都输入你的 MATLAB 命令窗口:

```
clear A
for i = 1:8
   A(i) = i;
end
A
```

```
clear A
parfor i = 1:8
   A(i) = i;
end
A
```

注意 A 的每个元素都与其索引相等。parfor 有效，因为每个元素只依赖于自己的迭代，不依赖其他的迭代。仅仅是重复这样独立任务的 for 循环很适合转换为 parfor 循环。

2.1.4 for-Loops 与 parfor-Loops 的差异

由于 parfor 循环和 for 循环不太一样，我们应了解其特别的行为。正如在前面的例子中看到的，当你在循环体内以循环变量作为索引向数组变量(例子中的 A)赋值，循环结束后数组元素是可用的，和 for 循环一样。

但是，假设你在循环体内使用非索引变量或者其索引不依赖循环变量 i 的变量。试试这些例子，注意循环体之后 d 和 i 的值:

```
clear A
d = 0; i = 0;
for i = 1:4
   d = i*2;
   A(i) = d;
end
A         [2,4,6,8]
d          8
i          4
```

```
clear A
d = 0; i = 0;
parfor i = 1:4
   d = i*2;
   A(i) = d;
end
A       [2,4,6,8]
d          0
i          0
```

注：此处所获得结果运行环境：intel 单核处理器，matlab r2009b

尽管两个例子中 A 的元素最后都是一样的，d 的值却不一定。在左边的 for 循环里，迭代时顺序执行，因此后面 d 保持着最后一次迭代后自己的值。在右边的 parfor 循环里，迭代时并行执行，不是顺序执行，因此不可能在循环结束时给 d 一个确定的值。这对循环变量 i 也适用。所以，parfor 循环的行为被定义为它不在循环外影响 d 和 i 的值，它们的值在循环之前和循环结束以后是一样的。因此，parfor 循环要求每次迭代都独立于其他迭代，而且 parfor 循环之后的代码不依赖迭代顺序。

2.1.5 规约赋值

下面的两个例子展示了 parfor 循环使用归约赋值(reduction assignments)。归约(reduction)就是跨过了循环迭代的累积过程。左边的例子用 x 通过循环的 10 次迭代积累一个和。右边的例子产生了一个连接数组，1:10。在两个例子中，迭代执行的顺序并不重要:当 workers 计算单独的结果时，client 适当地组合了最终的循环结果。

```
x = 0;
parfor i = 1:10
   x = x + i;
end
x
55
```

```
x2 = [];
n = 10;
parfor i = 1:n
   x2 = [x2, i];
end
x2
[1,2,3,4,5,6,7,8,9,10]
```

如果循环迭代以随机顺序操作，你可能觉得右边例子的连接顺序会是不连续的。但是，MATLAB 识别了连接操作并产生了确定的结果。

下面的例子试图计算 Fibonacci 数，它不是有效的 parfor 循环，因为 f 的迭代中一个元素的值依赖于 f 在其他迭代中计算的元素的值。

```
f = zeros(1,50);
f(1) = 1;
f(2) = 2;
parfor n = 3:50
    f(n) = f(n-1) + f(n-2);
end
```

当你结束了你的循环例子，清空你的工作区并关闭或者释放你的 pool of workers:

```
Clear
matlabpool close
```

接下来的部分提供更多的有关程序设计考量和 parfor 循环的局限的信息。

2.2 parfor 程序设计注意事项

2.2.1 MATLAB 路径

执行 parfor 循环的所有 workers 都必须有和 client 相同的 MATLAB 路径配置，这样它们就可以在循环体内调用任何函数。所以，每当你在 client 使用 cd，addpath 或者 rmpath，如果可能，(这些命令)也会在 workers 上面执行。如果你需要更多信息，参见 matlabpool 参考页。当 workers 运行在与 client 不同的平台，则使用 pactRunOnAll 为所有 workers 适当地设置 MATLAB 路径。

2.2.2 错误处理

当 parfor 循环在执行时发生了错误，所有的运行中的迭代都会终止，新的迭代不会被初始化，然后循环终止。

Workers 上产生的错误和警告会以 worker ID 作为注解并以被 client MATLAB 接收的顺序显示在 client 的命令窗口中。

lastwarn 和 lasterror 如果是在循环内使用，那么它们在 parfor 结束时的行为未知。

2.2.3 局限性

2.2.3.1 变量名含义模糊

当读取代码的时候，parfor 循环里不可以有模棱两可的名字(到底是变量还是函数？)。(参见 MATLAB 文档中 Naming Variables)。例如，在下面的代码中，当读取代码时，如果 f 既不是一个在路径中的函数，也没有在代码中清楚地定义，那么 f(5)可以指的是数组 f 的第五个元素，或者以 5 为参数的函数 f。

```
parfor i=1:n
    ...
    a = f(5);
    ...
End
```

2.2.3.2 透明性

parfor 循环体必须是透明的，意思是所有变量的引用必须“可见”(亦即，它们在程序的文本里出现)。

下面的例子里，由于 X 在 parfor 体内作为一个输入变量不可见(只有字符串'X'被传递给 eval)，它不能被传给 workers。结果 MATLAB 在运行时报错:

```
X = 5;
parfor ii = 1:4
    eval('X');
end
```

相似地，你不能在 parfor 语句内部使用 clear 从一个 worker 的工作区清除变量:

```
parfor ii= 1:4
    <statements...>
    clear('X')  % cannot clear: transparency violation
    <statements...>
End
```

作为一种变通，假设你在 parfor 语句里不再需要一个变量，你可以通过设置其值为空来释放其占有的大部分内存:

```
parfor ii= 1:4
    <statements...>
    X = [];
    <statements...>
End
```

一些函数违背了透明规则，例如 evalc，evalin 和与指定参数'caller'的 workspace 一起的 assignin；save 和 load，除非 load 的输出是给定的(assigned)。

当 eval 和 evalc 语句出现在 parfor 内调用的函数里时，MATLAB 则可以执行。

2.2.3.3 不可分函数

如果你在一个 parfor 循环或者由 parfor 循环调用的函数里使用一个性质上不是严格计算(strictly computational)的函数(例如，input，plot，keyboard)，这函数作用在 worker 上。结果可能包括挂起 worker 进程或者根本得不到可见效果。

2.2.3.4 嵌套函数

parfor 循环体不可以使用(make reference to)nested function。但是它可以以 function handle 的形式调用 nested function。

2.2.3.5 嵌套 parfor-Loops

Parfor 循环不可以包含另一个 parfor 循环。但是可以调用一个包含 parfor 循环的函数。

2.2.3.6 Break 和 Return 语句

Parfor 循环体不可以包含 `break` 和 `return` 语句。

2.2.3.7 全局一致变量

Parfor 循环不可包含 `global` 变量声明和 `persistent` 变量声明。

2.2.4 性能考虑

2.2.4.1 划分数组

如果一个变量在 parfor 循环前初始化并在 parfor 循环内使用，它就被发给每个 MATLAB worker 计算循环迭代。只有在循环内部使用的变量被 client 工作区发出去。但是，如果变量的所有出现都由循环变量索引，每个 worker 只接收它需要的数组的部分。更多信息，参见 `Where to Create Arrays`。

2.2.4.2 本地 vs. 群机

在本地 workers 上运行自己的代码可以为测试应用程序提供了方便，不需要使用集群资源。但是使用本地 workers 有固有的缺点或局限。因为数据不是通过网络传输，本地 worker 的传输可能不能表示网络传输情况。更多细节参见 `Optimizing on Local vs. Cluster Workers`.

2.2.5 与早期 MATLAB 软件的兼容性

在 7.5(R2007b)以前版本的 MATLAB 中，关键字 parfor 被定为 parfor 循环的另一种形式，比 7.5 或 7.5 以后的版本限制更多。原本打算在并行作业内部用旧的 parfor 循环与 codistributed arrays 一起使用，后来被使用 drange 定义范围的 for 循环取代。参见 Using a for-Loop Over a Distributed Range (for-drange).

关键字 parfor 过去和现在的功能描述如下表:

Functionality	Syntax Prior to MATLAB 7.5	Current Syntax
Parallel loop for codistributed arrays inside a parallel job	parfor i = range loop body . . end	for i = drange(range) loop body . . End
Parallel loop for implicit distribution of work	Not Implemented	parfor i = range loop body . . End

3 任务并行编程（spmd）

3.1 使用 spmd 结构

3.1.1 简介

单程序多数据（spmd）语言允许无缝的进行串行和并行交叉编程。Spmd 语句让你定义一个代码块并同时运行在多个 lab 上。在 lab 上 spmd 语句中赋值的变量允许直接在 client 上通过 composite 对象访问他们的值。

本章解释了 spmd 语句和 composite 对象的一些性质。

3.1.2 何时使用 spmd

Spmd 中的“Single program”方面指的是同一段代码运行在不同的多个 lab 上。你在一个 Matlab 客户端上运行一个程序，被标志为 spmd 模块的其他部分运行在其他 lab 上。当这些块运行完毕后，你的程序继续在客户端运行。

“Multiple data”方面指的是虽然 spmd 语句在所有的 lab 上运行相同的代码，但每一个 lab 可以有不同的，独有的数据。所以多数据集可以在多个 lab 上同时被容纳。

Spmd 使用的典型场景是需要在多个数据集上同时执行一个程序，在不同的 lab 间需要

交互和同步。一些常见的场景如下：

- 程序花费很长的时间来执行 - spmd 让几个 lab 同时计算结果。
- 程序运行在大型数据集上 - spmd 让这些数据分布在多个 lab 上。

3.1.3 使用 matlabpool 创建 matlab 资源

你使用 matlabpool 函数存储一定数量的 MATLAB lab(workers)来执行一个后来的 spmd 语句或者 parfor 循环。依赖于你的调度程序，labs 可能远程运行于一个机组上，或者他们可能运行于本地 Matlab 客户端。你通过选择并发配置来确定一个调度程序和机组。对于怎样管理和使用配置的描述，可以看 Programming with User Configurations.

开始这一节的例子，为你的 spmd 语句的赋值分配当地 Matlab labs.

matlabpool

这个命令打开了一定数量的的 Matlab 工作者会话，这个数量是由缺省并行配置决定的。如果本地配置是你的缺省值，并且没有制定工作者的数量，在你本地的 Matlab 客户端机器上将在每一个 core（最大 8 个）上开始一个工作者。

如果你不想使用缺省值，你可以在 matlab 语句中制定配置或者 labs 怎样使用。举个例子，你的缺省配置只是使用 3 个 labs，键入：

Matlabpool 3

为了使用不同的配置 ，键入：

Matlabpool MyconfigName

为了查询你现在是否有一个 Matlabpool 是打开的，键入：

Matlabpool size

这个命令返回一个值指明当前的池中 lab 的数量。如果命令返回 0，表明现在没有池是打开的。

注意：你的已经安装了并行计算工具包软件，如果没有池打开，spmd 语句在本地 Matlab 客户端运将行没有任何的并行执行。换句话说，它运行在你的客户会话里就像它是一个单独的 lab。

当你使用一个 MATLAB 池完毕，使用这个命令关闭它：

matlabpool close

3.1.4 定义一个 spmd 语句

Spmd 语句的一般形式是：

```
Spmd
    <statements>
End
```

在<statements>中描述的代码块会在 matlabpool 中在所有的 lab 上同时并行的执行。你想限制执行只使用这些 lab 中的一部分，准确的指定运行多少个 lab 即可：

```
Spmd(n)
    <statements>
End
```

语句要求 n 个 lab 运行 spmd 代码，n 必须小于等于在打开 Matlab 池中 lab 的数量。如果池足够大，但是 n 个 lab 并不可用，语句将等待直到 lab 可用。如果 n 为 0，spmd 语句没有使用 lab，只是在本地客户端运行，就像当时没有池是开着的。

你可以指定 lab 数量的范围

```
Spmd(m,n)
    <statesments>
End
```

在这种情况下，spmd 语句需要最小为 m 个 lab，它使用最大为 n 个 lab。

如果控制执行 spmd 语句的 lab 数量是重要的，需要在你的配置中或者在 spmd 语句中准确的设置这个数字，而不仅仅是使用范围。

据个例子，在 3 个 lab 上创建一个随机矩阵：

```
matlabpool
spmd（3）
    R=rand（4，4）;
end
matlabpool close
```

注意：本章所有的接下来的例子都认为 MATLAB 池是开着的，并且在 spmd 语句之间保持开放状态。

不像 parfor 循环，每一个 spmd 语句中使用的 lab 每一个都有一个独一无二的值 labindex。这个让你指定代码使之只运行在某些 lab 上，或者指定执行，通常是为了访问唯一的数据。

据个例子，依赖 labindex 创建不同大小的数组。

```
Spmd（3）
    if labindex ==1
        R= rand（9，9）;
    else
        R=rand（4，4）;
    end
```

根据 labindex 来从每一个 lab 上下载唯一的数据，在每一个 lab 上使用相同的函数计算

出一个结果。

```
spmd（3）
    labdata=load（‘datafile_’  num2str(labindex) ‘.ascii’）
    result=MyFunction(labdata)
end
```

执行了一个 spmd 语句的 lab 同时操作并且知道彼此。正如一个并行的工作，你可以直接控制不同 lab 间的通信，在他们之间传输数据，使用分布式数组。

对于一系列可以简化这种能力的工具箱函数，参见函数部分

Interlab Communication Within *a* Parallel Job and Codistributed Arrays.

举例，在一个 spmd 语句中使用分布式数组：

```
spmd (3)
    RR = rand(30, codistributor());
end
```

每一个 lab 有一个 30*10 段的分布式数组。想要看关于分布式数组的更多信息，参见 *Math with Codistributed Arrays.*

3.2 通过 Composites 访问数据

3.2.1 简介

在 MATLAB 客户端会话的 composite 对象让你直接在 lab 上获取数据值。多数情况下在 spmd 语句中分配这些变量。在他们的显示和使用中，Composites 类似于单元数组。有两种方法来创建 Composite：

- 在客户端使用 Composite 函数。分配给该 Composites 元素的值存储在 lab 中。
- 定义变量在 lab 上 spmd 语句中。Spmd 语句中，作为 Composites，存储的值在客户端是可以被访问。

3.2.2 在 spmd 语句中创建 composite

当您定义或在一个 spmd 语句内为变量分配了值，数据值便存储在 lab 上。

在 spmd 语句后，这些数据值作为 Composite 可以在客户端被访问。Composite 对象就像单元数组，表现的很相似。在客户端，每个 Composite 为每个 lab 分配元素。例如，假设您打开一个有三个本地工作者的 Matlab 池，并且在池中运行 spmd 语句：

```
matlabpool open local 3

spmd    % Uses all 3 workers
    MM = magic(labindex+2); % MM is a variable on each lab
end
```

```
MM{1} % In the client, MM is a Composite with one element per lab
     8     1     6
     3     5     7
     4     9     2

MM{2}
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

一个变量可能无法定义在每个 lab 上。有关该变量没有定义的 lab，相应的 Composite 元素没有值。试图读取该元素将抛出一个错误。

```
spmd
    if labindex > 1
        HH = rand(4)
    end
end
HH
    1: No data
    2: class = double, size = [4    4]
    3: class = double, size = [4    4]
```

你也可以在从客户端设置 Composite 元素的值。这将引起数据的传输，把值存储在合适的 lab 上，即使在 spmd 语句中并不执行：

```
    MM{3} = eye(4);
```

在这种情况下，MM 必须已经作为一个 Composite 存在，否则 Matlab 将其理解为一个单元数组。

现在当你进入一个 spmd 语句，在 lab3 上变量 MM 的值被设置为：

```
spmd
    if labindex == 3, MM, end
end
Lab 3:
    MM =
         1     0     0     0
         0     1     0     0
         0     0     1     0
         0     0     0     1
```

当你在客户端作业空间使用 Composite 元素明确的赋值一个变量，数据从 lab 传到客户端

```
M = MM{1} % Transfer data from lab 1 to variable M on the client
     8     1     6
```

3 5 7
4 9 2

将一个完整的 Composite 赋给另一个 Composite 并不引起数据传递。换而言之，客户端仅仅是赋值 Composite 作为一个对 lab 上某些数据的引用：

NN = MM % Set entire Composite equal to another, without transfer

然而，访问一个 Composite 元素，来赋值给其他的 Composites，会导致数据从 lab 传输到客户端，即使这个数据是赋给同样的 lab。在这种情况下，NN 必须已经作为一个 Composite 存在。

NN{1} = MM{1} % Transfer data to the client and then to lab

当结束的时候，你可以关闭池：

matlabpool close

3.2.3 变量的持久性和 spmd 的次序

存在 lab 上的值在 spmd 语句之间依然有效。这允许你顺次的使用多个 spmd 语句，并且继续使用在 spmd 块中定义的相同的变量。

值依然保存在 lab 上直到相关的 Composites 在客户端被清理掉了，或者 Matlab 池被关闭了。下面的例子说明了在 spmd 块中数字的值的生存期，使用一个有 4 个工作者的池。

```
matlabpool open local 4
spmd
    AA = labindex; % Initial setting
end
AA(:)   % Composite
    [1]
    [2]
    [3]
    [4]
spmd
    AA = AA * 2; % Multiply existing value
end
AA(:)   % Composite
    [2]
    [4]
    [6]
    [8]
clear AA % Clearing in client also clears on labs
spmd; AA = AA * 2; end      % Generates error
matlabpool close
```

3.2.4 在 spmd 外创建 composite

Composite 函数不使用一个 spmd 语句，也可以创建 Composite 对象。这可能对于 lab 上可重入变量值有用，在一个 spmd 语句开始在这些 lab 上执行之前。假设一个 MATLAB 池已经打开了：

```
PP = Composite()
```

缺省情况下，创建一个 Composite，在 matlab 池中为每一个 lab 分一个元素。你也可以创建 Composite 仅仅在池中的 lab 的几个子集上。参见 Composite 引文获取更多的细节。Composite 的元素现在可以在客户端上像平时一样设置，或者像在 spmd 语句中的变量一样。当你设置 Composite 中的元素的时候，数据立刻被传递到相应的 lab 中：

```
for ii = 1:numel(PP)
    PP{ii} = ii;
End
```

3.3 分布式数组

3.3.1 分布式 vs codistrbuted 数组

您可以在 MATLAB 客户端创建一个数组，其数据存储在开放的 MATLAB 池的 lab 中。分布式数组分布在一维，沿着最后非单独维，并尽可能均匀地沿着 lab 间的这一维。当创建一个分布式的阵列的时候你无法控制这种分布的细节。

您可以在 lab 自己上创建一个 codistributed 数组，或者在一个 spmd 语句中，在 pmode 模式下，或者在一个并行作业中。当创建 codistributed 数组时，您可以控制分布的各个方面，包括维度或者分配。

分布式数组和 codistributed 数组之间的关系是关注点之一。Codistributed 数组被分割在 lab 之间中，在其中您执行代码来创建或操纵他们。分布式数组被分割在客户端之间的 lab 上，使用开放的 MATLAB 池。当您在客户端创建分布式数组，您可以在 spmd 语句内作为一个 codistributed 数组访问它。当您在一个 spmd 语句中创建一个 codistributed 数组时，您可以在客户端作为一个 distributed 数组访问它。只有 spmd 语句让您访从两个不同的地方访问相同的数组数据。

3.3.2 创建分布式数组

你可以使用几种方式中的来创建一个分布式数组：

- 使用 distributed 函数来在一个存在的数组从客户端工作区分配到开放的 matlab 池中的 lab 上。
- 使用过载的 distributed 对象方式来直接在 lab 上创造一个分布式数组。这种方式不需要数组已经在 lab 上存在。这些过载的函数包括 distributed.eye ,distributed.rand 等等。看完整的列表，参见 distributed 对象参考页。
- 在 spmd 生命中创建一个 codistributed 数组，然后在 spmd 语句外面像分布式数组

一样访问它。这使你使用分布式方案而不是缺省方式。

以上前两种方式在拆创建数组时并不涉及到 spmd。但是你可以看到 spmd 可能怎样操控这样被创建的数组：举例如下：

在客户端工作区创建一个数组，然后使它变成一个分布式数组：

```
matlabpool open local 2
W = ones(6,6);
W = distributed(W); % Distribute to the labs
spmd
T = W*2; % Calculation performed on labs, in parallel.
% T and W are both codistributed arrays here.
end
T % View results in client.
% T and W are both distributed arrays here.
matlabpool close
```

3.3.3 创建 codistributed 数组

你可以使用几种方式中的任何一种来创建一个 codistributed 数组：

- 在一个 spmd 语句中，一个并行作业或者 pmode 使用 codistributed 函数来分配已经在 lab 上运行那个作业的数据。
- 使用过载的 codistributed 对象方式来直接在 lab 上创造一个 codistributed 数组。这种方式不需要数组已经在 lab 上存在。这些过载的函数包括 codistributed.eye ,codistributed.rand 等等。看完整的列表，参见 codistributed 对象参考页。
- 在 spmd 语句外创建一个分布式数组，然后在运行在相同的 MATLAB 池中的 spmd 语句中像 codistributed 数组一样访问它。

在这个例子中，你在一个 spmd 语句中使用一个非缺省方式的分布式方案创建一个 codistributed 数组。首先 创建一个在第三维上的 1-D 的分布，其中有 4 部分在 lab1 上，12 个部分在 lab2 上。然后创建一个 3*3*16 的数组，数组内容值为 0。

```
matlabpool open local 2
spmd
codist = codistributor1d(3, [4, 12]);
Z = codistributed.zeros(3, 3, 16, codist);
Z = Z + labindex;
end
Z % View results in client.
% Z is a distributed array here.
matlabpool close
```

关于 codistributed 数组的更多细节，看第五章”Math withCodistributed Arrays”和第四章, “Interactive Parallel Computation with pmode”.

3.4 编程建议

3.4.1 MATLAB 路径

所有的 lab 中的 spmd 的执行语句必须与客户端有相同的 MATLAB 路径，这样它们可以执行在他们的通用代码块中调用的任何函数。因此，无论何时在客户端你使用 cd，addpath 或者 rmpath，它也在所有的 lab 上执行，如果可能的话。更多的信息，参见 matlabpool 参考页。当 lab 运行在不同的平台上除了客户端，使用 pctRunOnAll 函数在所有的 lab 上恰当的设置 MATLAB 路径

3.4.2 错误处理

当错误发生在执行一个 spmd 语句时，错误被报告给客户端。客户端试图在所有的 lab 上打断执行，然后抛出一个错误给用户。

Lab 上制造的错误和警告被 labID 标注，显示在客户端命令行窗口他们从客户端收到的命令。

3.4.3 局限性

透明性

Spmd语句的必须是透明的，意味着所有的关于变量的应用都是可见的*(i.e., they occur in the text of theprogram)*.

在下面的例子中，因为作为一个输入变量 X 在 spmd 体中是不可见的（只用字符串'X'被传到 eval 中），对于 lab 来说并不是透明的。因此，MATLAB 在运行时会出现一个错误。

```
X = 5;
spmd
eval('X');
end
```

类似的，你不能在一个 spmd 语句中通过执行 clear 来从一个工作者空间中删除一个变量

```
spmd; clear('X'); end
```

要想从一个工作者中删除一个具体的变量，从客户端工作区中删除它的 composite。作为选择，你可以通过将其值设置为空的方式，释放一个变量使用的尽可能多的内存，当他大概在你的 spmd 语句中不再被需要的时候。

```
spmd
<statements....>
X = [];
End
```

一些其他违背透明性原则的函数的例子为：evalc,evalin, assignin 当工作区参数被设置为 caller，save 和 load 的时候，如果 load 的输出不被赋值。

MATLAB 成功的执行出现在 spmd 体中被调用的函数中的 eval 和 evalc 语句。

嵌套函数

在一个函数里，spmd 语句体不能直接引用一个嵌套函数。然而它可以通过一个被定义为处理嵌套函数的函数的变量来调用一个嵌套函数。

因为 spmd 体在工作者中执行，被 spmd 体中调用的嵌套函数改变的变量并不在外函数的工作区中被改变。

异步函数

Spmd 语句体并不能定义异步函数，然而，它能够通过函数句柄的方式来引用一个异步函数

嵌套 spmd 语句

Spmd 语句体内并不能包含其他的 spmd。然而，它可以调用一个函数包含其他地 spmd 语句。保证你的 matlab 池有足够的空间来容纳这样的扩展。

嵌套 parfor 循环

一个 parfor 循环内并不能包含一个 spmd 语句，反之亦然。

break 和 return 语句

spmd 语句中不能包含一个 bread 和 return 语句。

全局和持久变量

Spmd 语句中不能包含一个全局的或者持久变量。

4 多核环境下 matlab 并行工具箱运行方法

要运行并行程序(所有的并行程序都要这么做)，首先要

```
matlabpool open local lab_num
```

lab_num 是你要打开的 lab(也叫 worker，我理解为为我们做计算工作的并行进程)数量(不能超过 CPU 核心数，所以其数量代表并行度)，也可以不指定 lab_num:

```
matlabpool
```

默认打开的 lab_num 数量与 CPU 核心数相同。不执行这条命令，后面的工作也可以进行，但那就不是并行了。

运行结束后请

```
matlabpool close
```

关闭打开的 labs.

5 矩阵向量乘与矩阵矩阵乘多核并行化

我写了 4 个函数(函数定义以及注释见相关 matlab 文件):

single_row_multiply_matrix(row_vector, matrix);	行向量与矩阵相乘
matrix_multiply_for(matrix_a, matrix_b);	矩阵乘(使用 for 循环)
matrix_multiply_parfor(matrix_a, matrix_b);	矩阵乘(使用 parfor 循环)
contrast_parfor_matrix_multiply(matrix_a, matrix_b);	计算矩阵乘时 parfor 相对 for 加速比

注:矩阵乘函数一样可以用于矩阵乘以列向量——此时 matrix_b 应为列向量。

串行代码	for i = 1:am
	  res(i,:) =
	  single_row_multiply_matrix(matrix_a(i,:),matrix_b);
	end

并行代码	parfor i = 1:am
	  %seperate matrix_a to rows,parallel on row level
	  res(i,:) =
	  single_row_multiply_matrix(matrix_a(i,:),matrix_b);
	end

其中 single_row_multiply_matrix(row,matrix)函数计算一个行向量与一个矩阵相乘。

parfor 性能测试结果如下(对于同一数据规模和 lab 数量，测试多次，取较稳定的结果)：

lab 数量\加速比\矩阵规模	100x100 * 100x100	200x200 * 200x200	1000x1000 * 1000x1000
1	0.6338	0.898	0.9917
2	0.7427	1.5654	1.9597
3	0.7552	2.0552	2.85
4	0.6303	2.2043	3.7074
lab 数量\加速比\矩阵规模	200x200 * 200x1	800x800 * 800x1	2000x2000 * 2000x1
1	0.26	0.6187	0.6561
2	0.2075	0.9697	1.1426
3	0.1746	1.074	1.5306
4	0.13	1.069	1.7463

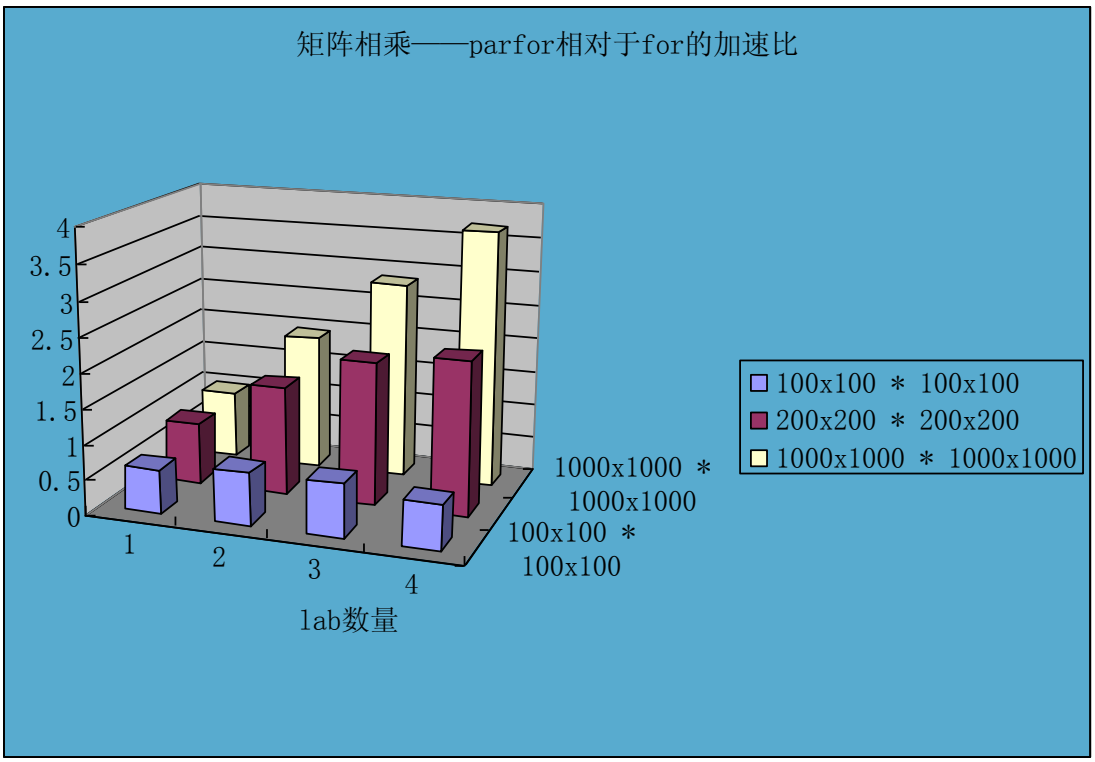


图 1-矩阵相乘中 parfor 相对 for 的加速比

选取了三个矩阵规模:100x100 * 100x100，200x200 * 200x200，1000x1000 * 1000x1000

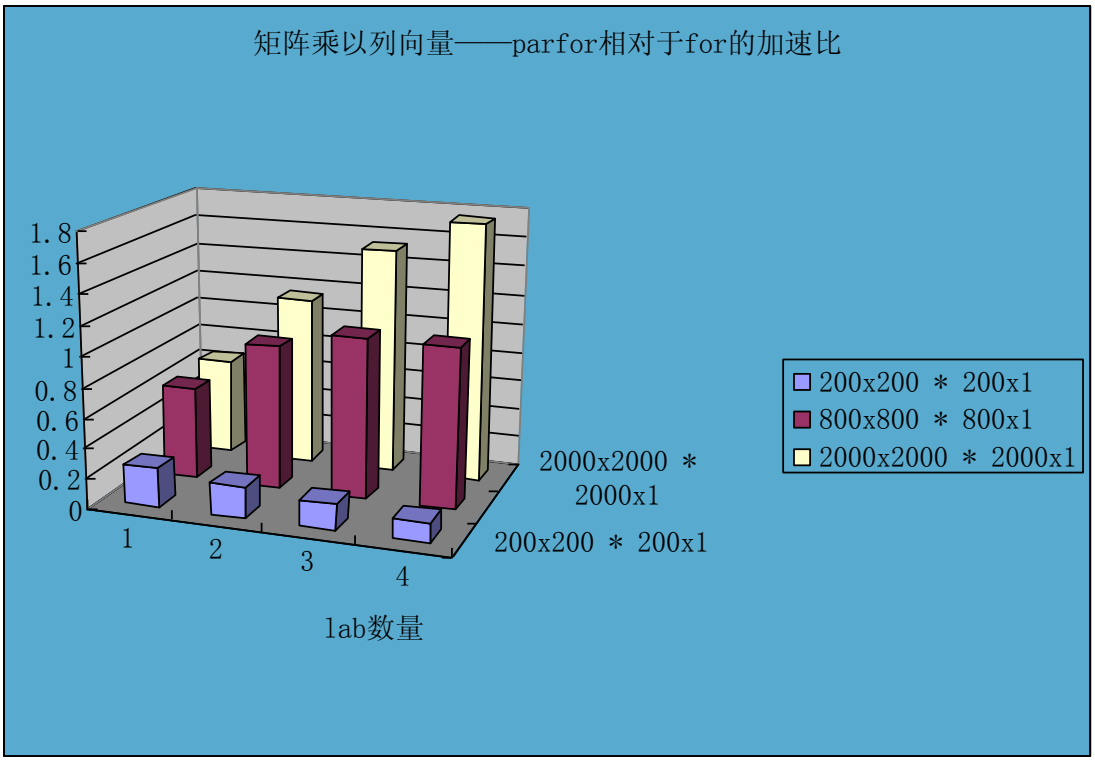


图 2-矩阵乘以列向量中 parfor 相对 for 的加速比

选取了 3 个计算规模:200x200 * 200x1，800x800 * 800x1，2000x2000 * 2000x1

从两幅图里都可以看到当计算规模较小时(100x100)parfor 加速不明显甚至性能比 for 还低，而且 lab 数量变大可能造成性能下降，这是因为将任务分配给 labs 会耗时，这种情况下 parfor 并行得来的性能提升有限，不能弥补这种消耗。矩阵规模较大(1000x1000)时加速明显，而且随着 lab 数量增加而增加，近似线性加速。