**Лабораторная 5**

**Использование MATLAB в параллельном режиме**

Необходимо выполнить данное ниже задание основываясь на методичке:

Дана последовательность натуральных чисел {*a0…an–1*}. Создать приложение для поиска минимального *ai*.

**Двумя способами.**

**1.** Выполнить решение задачи с помощью обмена сообщениями для любого числа процессов (при помощи MPI)

**2.** Выполнить решение с помощью spmd и обмена сообщениями для любого числа процессов

.

1. Передача сообщений между рабочими сеансами

В приложении PCT систе­мы MATLAB имеется встроенный набор функций для передачи сообщений между процессами, основанный на интерфей­се передачи сообщений Message Passing Interface (MPI).

**Посылка данных другому lab**

labSend(данные любого типа, номер процесса - получателя)

**Получение данных от другого lab**

данные = labReceive

данные = labReceive(номер lab)

Пример: суммирование элементов массива с подсчетом подсумм в отдельных рабочих процессах. Запускающий m-файл:

jm = findResource('scheduler','type','jobmanager','configuration','local')

pjob=jm.createParallelJob();

set(pjob,'MinimumNumberOfWorkers',3);

set(pjob,'MaximumNumberOfWorkers',3);

%x=rand(100,1)\*10;

x=[1 2 3 4 5 6 7 8 9 10];

set(pjob, 'FileDependencies', {'par\_sum.m'});

createTask(pjob,'par\_sum',1,{x});

submit(pjob); waitForState(pjob);

results = getAllOutputArguments(pjob)

destroy(pjob)

s=0;

for i=1:length(x)

s=s+x(i);

end

s

Функция par\_sum в файле par\_sum.m:

function z=par\_sum(x)

z=0;

m=length(x);

if ( labindex == 1)

%DataInitialization(x,N);

end

% Вычисление частичной суммы на каждом из процессов

% на каждом процессе суммируются элементы вектора x от i1 до i2

k =fix(m / numlabs);

i1 = k \* (labindex-1)+1;

i2 = k \* ( labindex );

if ( labindex == numlabs )

i2 = m;

end

ProcSum=0;

for i = i1:i2

ProcSum = ProcSum + x(i);

end

% Сборка частичных сумм на процессе с номером 1

if ( labindex == 1 )

TotalSum = ProcSum;

for i=2:numlabs

ProcSum=labReceive(i);

TotalSum = TotalSum + ProcSum;

end

z=TotalSum;

else

% Все процессы отсылают свои частичные суммы

labSend(ProcSum, 1);

end

1. **Запуск одной программы во всех рабочих процессах**

SPMD (single program - multiple data, одна программа – много данных) позволяет незаметно чередовать обычное и параллельное программирование. Оператор spmd позволяет определить блок кода для запуска на многих рабочих процессах. Переменные, определенные внутри блока spmd на различных lab, можно использовать в клиентском процессе как объект Composite.

"Одна программа" означает, что один и тот же программный код запускается на различных рабочих процессах. Когда блок spmd завершается (по оператору end) программа продолжает свою работу на клиенте.

"много данных" означает, что хотя код блока spmd идентичен на всех lab, каждый рабочий процесс может использовать уникальные данные.

Типичные случаи применения spmd следующие:

* Программы, имеющие длительное время выполнения — spmd позволяет нескольким lab выполнять алгоритм одновременно.
* Программы, оперирующие с большими наборами данных — spmd позволяет разделить данные между многими рабочими процессами.

Перед запуском spmd нужно выполнить команду [matlabpool](jar:file:///C:/Program%20Files/MATLAB/R2009b/help/toolbox/distcomp/help.jar%21/matlabpool.html) для создания некоторого числа рабочих процессов MATLAB (lab) для выполнения блоков spmd. lab могут выполняться удаленно на кластере или локально на клиентской машине. Команда matlabpool запускает столько рабочих процессов, сколько их определено в конфигурации local.

Можно запустить команду вида matlabpool 3 для запуска конкретного числа рабочих процессов. Их число на локальном компьютере не более 8.

Когда заканчивается использование пула pool,он закрывается по команде

matlabpool close

### 

### Определение блока spmd имеет вид:

spmd

операторы

end

Например, создадим случайную матрицу:

matlabpool 3

spmd

R = rand(4,4);

end

matlabpool close

Каждый lab, использующий spmd, имеет уникальное значение [labindex](jar:file:///C:/Program%20Files/MATLAB/R2009b/help/toolbox/distcomp/help.jar%21/labindex.html). Это позволяет определить уникальный фрагмент кода для некоторых lab.

Например, создадим массивы различного размера в зависимости от labindex:

spmd

if labindex==1

R = rand(9,9);

else

R = rand(4,4);

end

end

Загрузим разные файлы в зависимости от abindex и используем одну и ту же функцию в каждом lab, обрабатывая разные данные:

spmd

labdata = load(['datafile\_' num2str(labindex) '.ascii'])

result = MyFunction(labdata)

end

Можно контролировать обмен между lab, передавать данные между ними использовать массивы codistributed между ними.

**Объекты Composite**

Объекты Composite в клиентской сессии позволяют напрямую иметь доступ к значениям данных в lab. Имеется два способа создать Composites:

* Использование функции [Composite](jar:file:///C:/Program%20Files/MATLAB/R2009b/help/toolbox/distcomp/help.jar%21/composite.composite.html) на клиенте. При этом значения, назначенные таким элементам, сохраняются в lab.
* Определение переменных внутри блока [spmd](jar:file:///C:/Program%20Files/MATLAB/R2009b/help/toolbox/distcomp/help.jar%21/spmd.html). После окончания блока spmd сохраненные значения доступны на клиенте как Composites.

Более употребителен второй способ.

На клиенте Composite имеет один элемент для каждого рабочего процесса. Например, предположим, что открыт пул из трех рабочих процессов и запущен блок spmd на этом пуле:

matlabpool 3

spmd % Использует все рабочие процессы

MM = magic(labindex+2); % MM – переменная каждого lab

end

MM{1} % На клиенте MM - Composite с одним элементом на 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

Переменная может быть неопределенной на каком-то lab. Для тех lab, на которых переменная не определена, соответствующий элемент Composite не имеет значения. Попытка прочитать этот элемент приводит к ошибке.

spmd

if labindex > 1

HH = rand(4)

end

end

На клиенте:

HH

1: No data

2: class = double, size = [4 4]

3: class = double, size = [4 4]

Еще пример:

spmd

if labindex > 1

BB = rand(labindex)

end

end

Lab 2:

BB =

0.3802 0.7370

0.1215 0.3625

Lab 3:

BB =

0.7895 0.3064 0.7922

0.3299 0.6448 0.8524

0.0525 0.7349 0.9092

На клиенте: BB

BB =

1: No data

2: class = double, size = [2 2]

3: class = double, size = [3 3]

Также можно задать значение элемента Composite на клиенте. При этом данные передаются на соответствующий lab, даже если он не выполняется в блоке spmd:

MM{3} = eye(4);

При этом переменная MM уже должна существовать как Composite

Функция [Composite](jar:file:///C:/Program%20Files/MATLAB/R2009b/help/toolbox/distcomp/help.jar%21/composite.composite.html) объекты Composite без использования блока spmd. Это может быть полезно для предварительного присвоения значений переменным на lab перед исполнением блока spmd. Например:

PP = Composite()

PP =

1: No data

2: No data

3: No data

for ii = 1:numel(PP)%numel – число элементов в Composite

PP{ii} = ii

end

>> PP{:}

ans = 1

ans = 2

ans = 3

Если на клиенте задано значение MM{3}, то выполним

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

Данные передаются из lab на клиента, если использовать переменную как Composite элемент:

M = MM{1} % Преобразование данных из lab 1 в переменную M на клиенте

8 1 6

3 5 7

4 9 2

Присвоение всего объекта Composite другому не приводит к передаче данных. Вместо этого клиент просто дублирует Composite как ссылку на соответствующие данные, сохраненные в lab:

NN = MM % Присвоение всего Composite, равного другому без передачи

Однако доступ к элементам Composite для присвоения значений другим Composite приводит к передаче данных от lab к клиенту. В этом случае переменная NN должна уже существовать как Composite:

NN{1} = MM{1} % Передача данных клиенту и затем lab

### В таблице показано совместное использование клиента и двух рабочих процессов:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Client | Worker 1 | Worker 2 |
|  | a b e| | c d f | c d f |
| a = 3; | 3- - | - - - | - - - |
| b = 4; | 3 4 - | - - - | | - - - |
| spmd |  |  |  |
| c = labindex(); | 3 4 - | 1 - - | 2 - - |
| d = c + a; | 3 4 - | 1 4 - | 2 5 - |
| end |  |  |  |
| e = a + d{1}; | 3 4 7 | 1 4 - | 2 5 - |
| c{2} = 5; | 3 4 7 | 1 4 - | 5 6 - |
| spmd |  |  |  |
| f = c \* b; | 3 4 7 | 1 4 4 | 5 6 20 |
| end |  |  |  |

### **Длительность существования переменной**

### **и последовательность блоков spmd**

Значения, сохраненные на lab, сохраняются между блоками spmd. Это позволяет использовать несколько блоков spmd и продолжать использовать переменные, определенные в предыдущих блоках spmd.

Значения сохраняются на lab, пока соответствующие объекты Composite не будут очищены на клиенте или пока пул не будет закрыт. Пример иллюстрирует продолжительность жизни данных при наличии нескольких блоков spmd:

matlabpool 4

spmd

AA = labindex; % Начальная установка

end

AA(:) % Composite

[1]

[2]

[3]

[4]

spmd

AA = AA \* 2; % Умножение существующего значения

end

AA(:) % Composite

[2]

[4]

[6]

[8]

clear AA % Очмстка на клиенте также очищает все lab

spmd

AA = AA \* 2; % Генерируется ошибка

end

matlabpool close

**Функции, которые могут использоваться в рабочих процессах и на клиенте**

# gcat Глобальное соединение строк

Xs = gcat(X) сцепляет variant массивы X из каждого lab по второму измерению (по столбцам). Результат распространяется во все lab.

Xs = gcat(X, номер измерения) сцепляет variant массивы X из каждого lab по указанному измерению.

## Пример с 4 labs

Xs = gcat(labindex)

Возвращает Xs = [1 2 3 4] во все четыре lab.

# gop глобальное выполнение операции по всем lab

res = gop(@F, x)  
res = gop(@F, x, targetlab)

## Аргументы

|  |  |
| --- | --- |
| F | Функция, которая действует во всех lab. |
| x | Аргумент для функции F, должен быть одной и той же переменной во всех lab, но может иметь различные значения. |
| res | Переменная для сохранения результата. |
| targetlab | Рабочий процесс, которому возвращается результат. |

res = gop(@F, x) вычисляет функцию F по значению x из каждого lab. Результат дублируется во все lab.

Функция F(x,y) должна принимать два аргумента одного типа и получать один результат того же типа, так что он может использоваться итеративно, то есть

F(F(x1,x2),F(x3,x4))

Функция F должна быть ассоциативна, то есть,

F(F(x1, x2), x3) = F(x1, F(x2, x3))

res = gop(@F, x, targetlab) выполняет вычисления и помещает результат в переменную res на lab, указываемый параметром targetlab. Во всех остальных lab переменная res устанавливается в значение [].

## Примеры

Вычисление суммы всех значений x во всех lab.

res = gop(@plus,x)

Нахождение максимального значения x по всем lab.

res = gop(@max,x)

Можно создать собственную функцию для запуска функцией gop:

matlabpool(4)

spmd

x=labindex

res=gop(@pr,x)

end

res{1}

matlabpool close

Файл pr.m

function z = pr(x,y)

z=x.\*y;

end

**Измерение времени выполнения фрагмента кода**

# Это пара команд tic, toc

tic; любые операторы; toc;

измеряет время выполнения операторов. Команда tic начинает отсчет времени, затем выполняются операторы и toc останавливает таймер, выводя на экран время выполнения в секундах.

tic; любые операторы; tElapsed=toc; делает то же самое, но запоминает время выполнения в переменной tElapsed.

tStart=tic; любые операторы; toc(tStart); делает то же самое, но позволяет запустить несколько счетчиков времени. Результат tic запоминается в переменной tStart, и затем и используем ту же переменную при вызове toc. Измерение времени происходит от точки вызова tic и связанной с ней toc и выводит время в секундах.

tStart=tic; любые операторы; tElapsed=toc(tStart); делает то же самое, что и предыдущий вариант, но дополнительно запоминает время выполнения в переменной tElapsed.

Пример. Изменение минимального и среднего времени вычисления суммы функций Бесселя:

REPS = 1000; minTime = Inf; nsum = 10;

tic;

for i=1:REPS

tStart = tic; total = 0;

for j=1:nsum,

total = total + besselj(j,REPS);

end

tElapsed = toc(tStart);

minTime = min(tElapsed, minTime);

end

averageTime = toc/REPS;