Paper 114-28

Parallel Processing on the Cheap: *Using Unix Pipes to Run SAS® Programs in Parallel*

Ted Conway, Ted Conway Consulting, Inc., Chicago, IL

ABSTRACT

A small, general-purpose Unix script is presented that allows SAS (and other) programs to be run in parallel when SAS Connect and other parallel processing ETL tools aren't available or may be overkill for the task at hand. The script will primarily be of interest to those using Base SAS to process large amounts of data on Unix.

INTRODUCTION

Got a Unix-based application that could benefit from parallel processing?

If you don't have access to SAS Connect or expensive ETL tools (and the prerequisite training!), you may be tempted to give up without trying – especially if you've seen some complicated attempts at home-grown parallelism.

But - believe it or not - there is a (fairly!) straightforward way.

DESIGN CONSIDERATIONS

In designing a parallel processing solution, you'll want to:

- Make sure users can easily specify the maximum number of concurrent processes (no sense unleashing 500 concurrent processes on 3 processors, unless you're into angry phone calls from irate Unix administrators!).
- Dispatch tasks dynamically as threads free up. While dividing up processes ahead of time into equal groups is indeed simpler and easier, you want to minimize the chance of winding up with all but one of your threads finishing earlier, leaving you with what is essentially sequential processing.

Oh yeah – just one more thing.

Make it as easy as possible to use!

PIPIN' HOT PARALLELISM

From the above, it's seen that the key element that's needed is some form of interprocess communication between dispatched child processes and the parallel processing parent script.

And that's where our trusty Unix pipes come into play!

Our parent script will:

- Create a named pipe (queue for my fellow mainframe CICS refugees!) at start-up that will be used for subsequent interprocess communications.
- Read a file of Unix commands (e.g., SAS program invocations) provided by the user and launch the specified maximum number of processes in the background.
- Dispatch new processes as earlier child processes notify the parent of their completion (and exit status) via writes to the named pipe.

A LOOK UNDER THE HOOD

The following is a version of a workable script that's been "stripped down" (e.g., error handling logic's been removed) to (hopefully) make the underlying concept a little clearer.

The script's two parameters consist of a user-specified file name containing a list of Unix commands and a value for the maximum number of concurrent processes.

"RunParallel" – Parallel Processing Script

```
# Script To Run A File ($1) Of Commands In ($2) Concurrent Parallel
Processes

pipen=$$
mkfifo completed.processes.named.pipe.$pipen
activeProcesses=0
processNumber=0

while read aCommand ; do
  let processNumber=$processNumber+1
  if [ $activeProcesses -ge $2 ] ; then
    completedProcesses=`cat completed.processes.named.pipe.$pipen`
    for completedProcess in $completedProcesses ; do
      let activeProcesses=$activeProcesses-1
    done
    fi
  let activeProcesses=$activeProcesses+1
  ( ksh $aCommand ; \
    print "$processNumber:$?" >> completed.processes.named.pipe.$pipen; ) &
done < $1

while [ $activeProcesses -gt 0 ] ; do
  completedProcesses=`cat completed.processes.named.pipe.$pipen`
  for completedProcess in $completedProcesses ; do
    let activeProcesses=$activeProcesses-1
  done
done

wait
rm completed.processes.named.pipe.$pipen
exit
```

AN EXAMPLE OF PARALLELISM IN ACTION

The following sections illustrate how run times for even a very trivial list of commands can be reduced substantially (from 120 to 35 seconds in this case!) by running them in parallel.

While the following uses a script (with a food theme!) to keep things simple, keep in mind that a list of SAS invocations (e.g., sas -nodms -sysparm "Burger For 5 Seconds" cook.sas) will work just the same!

"dinner" – User-Specified File With List Of Commands

```
$ cat dinner
cook HotDog      For   1 Seconds
cook Vegetables For   3 Seconds
cook Burger      For   5 Seconds
cook Steak       For  10 Seconds
cook Chicken     For  15 Seconds
cook Ribs        For  20 Seconds
cook Potato      For  30 Seconds
cook Roast       For  35 Seconds
```

"cook" – Testing Script

```
$ cat cook
echo Started Cooking $1 At `date +%H:%M:%S` For $3 $4
sleep $3
echo Finished Cooking $1 At `date +%H:%M:%S`
```

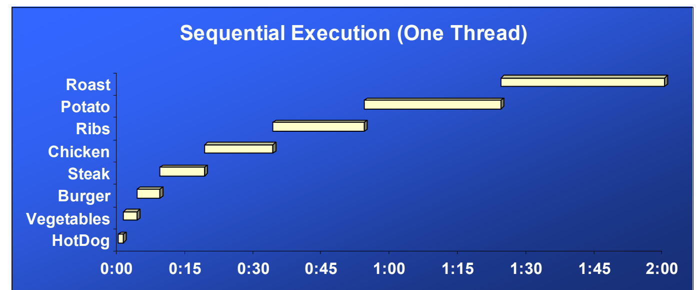
Sequential Processing (120 Seconds Elapsed Time)

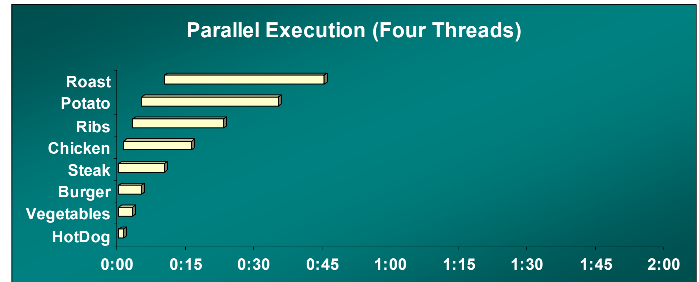
```
$ dinner
Started Cooking HotDog At 18:43:49 For 1 Seconds
Finished Cooking HotDog At 18:43:50
Started Cooking Vegetables At 18:43:50 For 3 Seconds
Finished Cooking Vegetables At 18:43:53
Started Cooking Burger At 18:43:53 For 5 Seconds
Finished Cooking Burger At 18:43:58
Started Cooking Steak At 18:43:58 For 10 Seconds
Finished Cooking Steak At 18:44:08
Started Cooking Chicken At 18:44:08 For 15 Seconds
Finished Cooking Chicken At 18:44:23
Started Cooking Ribs At 18:44:23 For 20 Seconds
Finished Cooking Ribs At 18:44:43
Started Cooking Potato At 18:44:43 For 30 Seconds
Finished Cooking Potato At 18:45:13
Started Cooking Roast At 18:45:14 For 35 Seconds
Finished Cooking Roast At 18:45:49
```

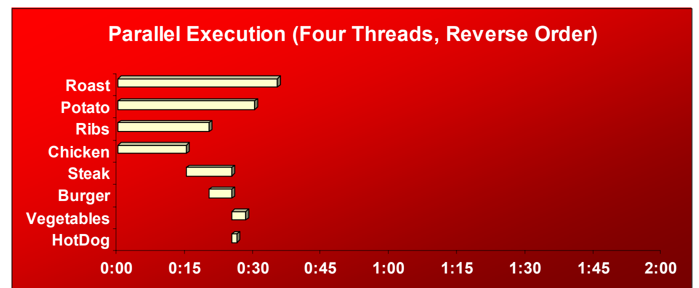
Parallel Processing (45 Seconds Elapsed Time)

```
$ RunParallel dinner 4
Started Cooking HotDog At 18:47:47 For 1 Seconds
Started Cooking Vegetables At 18:47:47 For 3 Seconds
Started Cooking Burger At 18:47:47 For 5 Seconds
Started Cooking Steak At 18:47:47 For 10 Seconds
Finished Cooking HotDog At 18:47:48
Started Cooking Chicken At 18:47:48 For 15 Seconds
Finished Cooking Vegetables At 18:47:50
Started Cooking Ribs At 18:47:50 For 20 Seconds
Finished Cooking Burger At 18:47:52
Started Cooking Potato At 18:47:52 For 30 Seconds
Finished Cooking Steak At 18:47:57
Started Cooking Roast At 18:47:57 For 35 Seconds
Finished Cooking Chicken At 18:48:03
Finished Cooking Ribs At 18:48:10
Finished Cooking Potato At 18:48:22
Finished Cooking Roast At 18:48:32
```

Sequential vs. Parallel Processing Performance







***Note:** This last chart illustrates how simply running the longest processes first can often increase overlap and boost performance.*

CONCLUSION

If you have excess processor capacity at your disposal and your data is organized and/or can be processed in 'convenient' groups (e.g., monthly files, area codes, states, account number ranges, etc.), dramatic reductions in run times can often be achieved by processing your data in parallel.

CONTACT INFORMATION

Ted Conway currently works for Ted Conway Consulting, Inc. (guess how he got that job!) in Chicago, Illinois. He can be reached at tedconway@aol.com.

TRADEMARKS

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.