Parallelization & Partitioning How-To

This document provides basic information on how to configure the various tools available within the TRANSIMS toolbox using parallelization and partitioning strategies with the goal of reducing model simulation runtime. The How-To specifically outlines a typical TRANSIMS assignment process using four central processing units (CPUs). The How-To assumes the reader has at least a working knowledge of the TRANSIMS tools. New users are encouraged to first test and experiment with the TRANSIMS toolbox before attempting to configure a parallelized model system.

Revision History

4/29/2010 Created by RSG, Inc.

Table of Contents

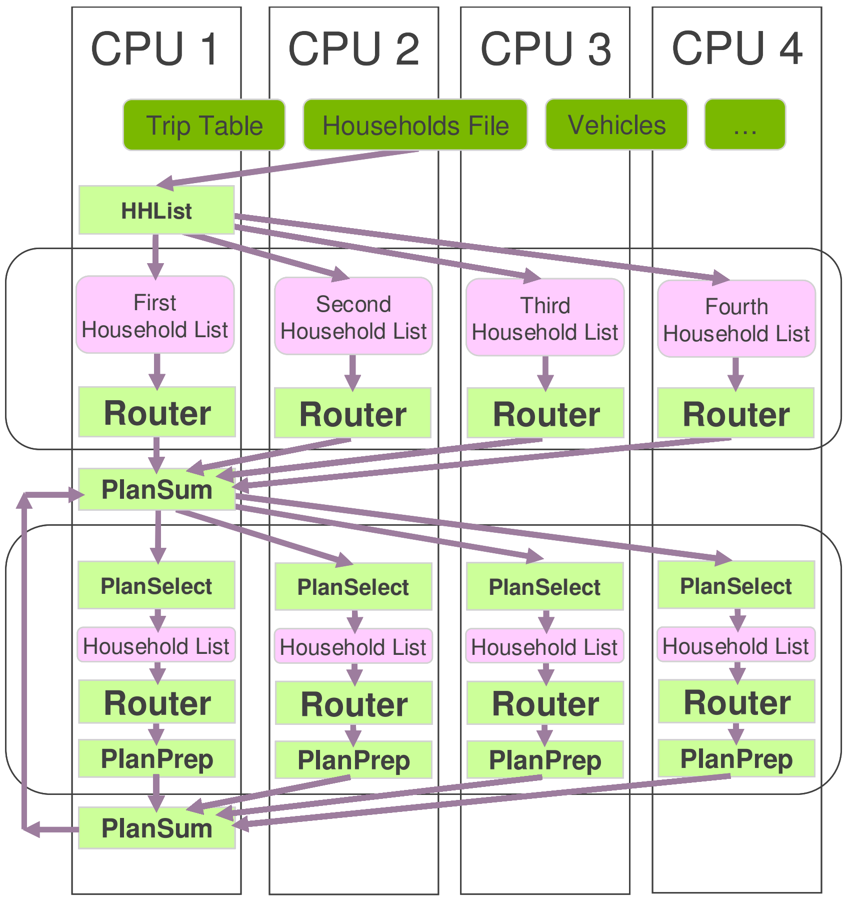
1.0 Introduction

Due to the disaggregate nature of TRANSIMS model systems which use very detailed spatial and temporal information, the model simulations are computationally intensive and can take a considerable amount of time to run when applied in very large regional simulations. In order to make the process more efficient, the TRANSIMS software can accommodate splitting the calculations into independent tasks that can be executed on different CPUs in a multi-processor desk machine or on a computer cluster. The results of these computations are then merged together to reflect the performance of the entire system. Partitioning makes the process feasible and time efficient, but it also creates increased complexity and implementation issues that are not encountered during a basic single CPU application.

Whether to implement a parallelization and partitioning scheme is a decision to be made by the research team. Performing a microsimulation for a large urban area will likely require the use of parallelization since runtimes would be unmanageable. The Burlington case study modeled a small urban region with approximately 420,000 daily vehicle trips, 60,000 households, and a network with 800 roadway links. A typical three-phase Router Stabilization, Microsimulator Stablization, and User Equilibrium model system was designed that performed 40 total iterations. This model system was run on a single CPU and took approximately 4 hours to run to completion, so parallelization was not necessary. On the other hand, the Moreno Valley case study modeled a very large region (Los Angeles metro area) with approximately 40 million daily vehicle trips and a network with 48,000 links. There, parallelization using 32 CPUs was necessary in order to obtain reasonable run times that still required roughly 90 hours for 80 iterations of the model system. New users are encouraged to familiarize themselves with the TRANSIMS toolbox and how to develop a model system before investigating the use of parallelization techniques.

Partitioning is based on the Router's capability to process a subset of trips and/or activities based on the household list which contains household ID numbers. Before running the Router a special utility called HHList is used to write random household subsets into simple household list files. The Router as well as other utilities such as PlanPrep, PlanSelect and so on will then only work on a smaller subset of the data files. PlanSum is run on the entire dataset to calculate link delays based on all the link volumes. Therefore, PlanSum is run on a single CPU so that all the plans calculated by the Router can be summarized.

The figure below illustrates how partitioning is applied to run the Router, PlanPrep, PlanSelect, and PlanSum.



The first step is to execute an initial run of the Router for the entire population as a starting point. The goal is to run four instances of the Router, each on a different subset of trips or activities. Trips and activities are grouped by households as the smallest inseparable unit. To begin, HHList is used to create four household lists which are populated randomly from the existing Households file. Each household in the Household file is present in exactly one of the four household lists. Each instance of the Router is started simultaneously, one on each of the four CPUs. Each instance of the Router reads the entire normal set of input files (shown in green above). However, the Router ignores trips and activities that do not belong to its assigned list of households (First, Second, Third, or Fourth) and writes a separate plan file for only its own plans. PlanSum is then used to evaluate all the plans, so it must be run on a single CPU. PlanSum reads the four partitioned plan files to create a single new link delay file.

To start additional iterations, PlanSelect is used to select travelers eligible for re-routing. PlanSelect can apply its criteria on just the partitioned plan file created by the initial routing step. The output from PlanSelect is a new household list for each partition which is simply a subset of the initial partitioned Household List (First, Second, Third, or Fourth). Because each new household list is independent, PlanSelect can likewise be run in parallel.

The Router is executed again and as before reads in all the normal input files and ignores trips and activities based on the Household List for each partition. In this case, the link delay file created by PlanSum in the first iteration is used to expose travelers to the latest congested travel times. The output is a plan file this time containing only a subset of plans for the travelers that were selected for re-routing.

The plans from the four partitions now only cover a subset of the trips selected for re-routing. Therefore, the new plans must be merged into each partition's master plan file, replacing the plan records from the previous iteration. This is accomplished using the PlanPrep tool which can also be applied in parallel since the master plan file is also partitioned. The result is a complete set of plans, still partitioned according to the original household lists, that are updated to reflect the most current iteration.

PlanSum is then executed again on a single CPU to generate a new link delay file from the complete set of updated , partitioned regional travel plans. This process would be iterated until the analyst's desired stopping criteria are achieved. The PlanSum portion of the iteration cannot be parallelized, so efficient use of computing time involves a trade-off between Router and PlanSun runtimes since the other CPUs remain idle while one CPU is running PlanSum.

The sample illustrates the use of four CPUs. However, TRANSIMS will accommodate up to 200 partitions. In most simulations of large urban areas, between 4 and 32 partitions has been demonstrated to yield acceptable runtimes depending on the amount of demand and geographic scale of the region being modeled. A parallelization process similar to the one illustrated above can be incorporated into the Router Stabilization, Microsimulator Stabilization, and Dynamic User Equilibirum phases, separately or in all cases. The goal is to reduce the amount of runtime required to run the Router to develop plans and subsequent iterative feedback when travelers are selected for re-routing.

2.0 HHList Utility

The HHList utility is needed to develop household lists that can be used to run the TRANSIMS tool in a parallelized manner. HHList is a very simple program that takes as input either a household list or a household file. It produces a user-specified number of household list files depending on the `NUM_SPLIT_FILES` control key parameter. The assignment of household IDs to partitioned household lists should be randomized for best load balancing across the CPUs; this is automatically handled by the HHList function. Since the ConvertTrips process may have grouped certain types of trips together which may have dramatically different characteristics, any other method of partition assignment may have adverse affects. For example, a user wouldn't want one CPU to process all the very long trips while another routed only short trips.

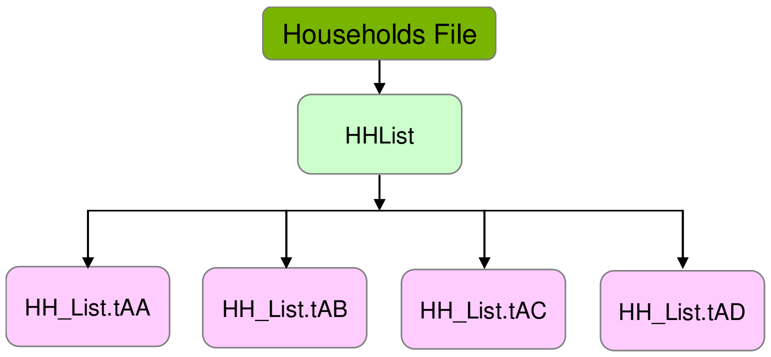
To execute the HHList utility a batch file can be created that contains the following:

HHList.exe HHList.ctl

A sample HHList.ctl control file is provided below:

```
TITLE                   Partition the Household File
DEFAULT_FILE_FORMAT     TAB_DELIMITED

#---- Input ----

HOUSEHOLD_FILE          Household_File
NUM_SPLIT_FILES         4
RANDOM_NUMBER_SEED      23456

#---- Output ----

NEW_HOUSEHOLD_LIST      HH_list
```

Executing the HHList utility in this fashion will create four household lists with the filenames HH_List.tAA, HH_List.tAB, HH_List.tAC, HH_List.tAD. If the Household_File has 200,000 households each HH_List.t* should have approximately 50,000 households. However, they will not all be equal in size given the random allocation. By specifying a RANDOM_NUMBER_SEED value parameter not equal to 0, the HHList.tAA households generated will always be the same every time it is executed. If the random number seed is not specified, the households in the .tAA file will be different each time the utility is executed because the program will generate a random number seed from the system clock.



3.0 Running Partitioned TRANSIMS Tools

The TRANSIMS toolbox contains 69 separate utilities. However, only 14 of the tools support parallelization and can be run in partitioned mode. The table below identifies which tools can be run in partitioned mode.

ActGen	PlanCompare	PlanTrips	SmoothPlans
AdjustPlans	PlanPrep	PopSyn	SubareaPlans
ArcPlan	PlanSelect	ProblemSelect	
EventSum	PlanSum	Router	

Some tools like the Router will be executed in parallel. That is to say, a user may have 4 instances of the Router running simultaneously on 4 different CPUs. Some tools, though, will be used only to read, process or manipulate partitioned plan files (e.g. PlanSum or PlanCompare). These tools are typically executed on a single CPU but read all the partitioned files sequentially (e.g. Plans.tAA, then Plans.tAB, then Plans.tAC, and finally Plans.tAD in this example).

The syntax for partitionable tools includes a simple integer at the end of the control list indicating the partition number for this process. For example, the syntax below starts the first partition (0-based integer):

```
Router.exe –B –K Router.ctl 0
```

Control files are also interpreted differently by the partitionable tools. File specifications for partitioned input and output files are internally extended by appending the extension ".tAA" to the names specified in the control file. "tAA" is appended for the first partition (partition 0), "tAB" is appended for the second partition (partition 1), "tAC" is appended for the third partition (partition 2) so on and so forth. The automatic appending means that control files can be identical for each partition and there is no requirement to write eight separate control files for use on eight separate partitions for example.

To start four instances of the Router simultaneously using a Windows batch file the syntax would be as follows:

```
start Router.exe –B –K Router.ctl 0
```

start Router.exe –B –K Router.ctl 1
start Router.exe –B –K Router.ctl 2
start Router.exe –B –K Router.ctl 3

A sample of the Router.ctl file is provided below. This assumes that the HHList utility has already been executed to create the HH_List.tAA, HH_List.tAB, HH_List.tAC, and HH_List.tAD partitioned household lists.

```
TITLE                               Route Selected Travelers

#---- Input Files ----

NET_NODE_TABLE                      Node
NET_LINK_TABLE                      Lane_Connectivity
NET_PARKING_TABLE                   Parking
NET_ACTIVITY_LOCATION_TABLE         Activity_Location
NET_PROCESS_LINK_TABLE              Process_Link

TRIP_FILE                           Trip
VEHICLE_FILE                        Vehicle
HOUSEHOLD_LIST                      HH_List

#---- Output Files ----

NEW_PROBLEM_FILE                    Problems
NEW_PLAN_FILE                       Plans
```

When the batch file is double-clicked, four instances of the Router will be started. Each instance will read a separate HH_List file. The Router will automatically use the HH_List.tAA households to process partition 0, and HH_List.tAB households to process partition 1, etc. even though they have not been specifically identified in the control file. Upon completion, four router print output files will be created: Router_0.prn, Router_1.prn, Router_2.prn, and Router_3.prn. Each print file summarizes the routing that was performed on that individual partition. In addition, four partitioned Plans output files will be created: Plans.tAA, Plans.tAB, Plans.tAC, Plans.tAD. Again, the file extensions are automatically appended using the integer codes specified in the batch file (0, 1, 2, 3, etc). Partitioned Problem files will also be created using the same naming convention rules.

The great benefit of running the Router in a parallelized manner is the routing will be completed roughly four times faster when the task is split among four CPUs compared to non-partitioned execution on a single CPU. That is to say, there is a roughly linear gain in performance as the number of processors is increased.

As described above, some of the tools in the TRANSIMS toolbox will not be run in a parallelized manner. But they are often used to read, write, or process partitioned files such as partitioned Plan files. One example of such a tool is the PlanSum utility. This utility is used to estimate link delays for the entire model network and therefore needs the link volumes from all of the travel plans for all households being modeled. This is accomplished by executing PlanSum on a single CPU, but configuring the PlanSum control file to read in all of the partitioned plan files. The Microsimulator is also usually executed on a single CPU since all regional travel plan files must be read and processed together to fully represent region-wide network conditions during the simulation.

To execute PlanSum using a Window batch file the syntax would be as follows:

```
PlanSum.exe –B –K PlanSum.ctl
```

A sample of the PlanSum.ctl file is provided below. Note that the PLAN_FILE is specified as Plans.t*. The wildcard .t* prompts the program to read in all four of the partitioned Plans files located in the working directory sequentially. Please note the program will read all the partitioned files present. Therefore, any old partitioned files that might remain from previous tests where a different number of partitions were utilized, for instance eight instead of four, should be removed to prevent corrupting the computations. An examination of the print out *.prn file will show '`Number of Plan Files = 4`'which is a useful check for ensuring how many partitioned files were actually read by the function.

```
TITLE                          PlanSum on Router Plans

# ---- Input Files ----

NET_LINK_TABLE                 Link
NET_ACTIVITY_LOCATION_TABLE    Activity_Location
NET_PARKING_TABLE              Parking

PLAN_FILE                      Plans.t*

# ---- Output Files ----

NEW_LINK_DELAY_FILE            Link_Delay

# ---- Parameters ----

SUMMARY_TIME_PERIODS           0..24
SUMMARY_TIME_INCREMENT         15

EQUATION_PARAMETERS_1          BPR, .15, 4.0, 1.
EQUATION_PARAMETERS_5          BPR, .15, 2.0, 1.0
```

If the user wants to process only a single partition file it can be specified with the exact extension. For instance if PLAN_FILE was specified as Plans.tAA only the .tAA partitioned plan file would be processed and the output *.prn file would show '`Number of Plan Files = 1`'

4.0 Programmatic Complexity of Partitioning

Developing a TRANSIMS model simulation system that takes advantage of the parallelization and partitioning capabilities provided by the software which processes data on multiple machines significantly increases the complexity of the model system.

Synchronization of all tasks in the model stream becomes essential. For instance, from our example above, PlanSum cannot be executed until all four instances of the Router have been completed successfully.

The coordination and control of multiple instances that have to function in a synchronized fashion must be carefully designed. Exit codes are explicitly written by each TRANSIMS utility to help the user write appropriate scripts to control model flow (0-success, 1-error, 2-warning).

Command line options have also been implemented to turn off potential interactive feedback.

"**-H**" explains the options available in the executables

"**-Q**" executes the tools with minimal screen messages and no interactive feedback

"**-B**" suppresses interactive feedback but writes progress and other screen messages

"**-K**" checks whether all control keys are actually supported

```
Router.exe –B –K Router.ctl
```

Executing the router in a batch file using the syntax above for instance will execute the Router by suppressing interactive feedback while also checking whether all control keys listed in the control file are supported. The –H command is very useful for determining which control keys are used by the particular function in use.

Let's say we want to run the Router and then PlanSum per the sample case illustrated above, but wanted to control the program flow using a single batch file that would perform both model steps as well as the household list partitioning.

```
HHList.exe –B –K HHList.ctl

Router.exe –B –K Router.ctl 0
Router.exe –B –K Router.ctl 1
Router.exe –B –K Router.ctl 2
Router.exe –B –K Router.ctl 3

PlanSum.exe –B –K PlanSum.ctl
```

The syntax above will start and finish the HHList execution. Next, the first instance of the Router would be launched, but the computer would then wait for partition 0 to finish before launching the Router again for partition 1. So on until PlanSum is run. This execution is fine, but it doesn't take advantage of the efficiency gains that can be achieved by running the Routers in parallel as opposed to in series. Consider the following syntax:

```
start Router.exe –B –K Router.ctl 0
start Router.exe –B –K Router.ctl 1
start Router.exe –B –K Router.ctl 2
start Router.exe –B –K Router.ctl 3

PlanSum.exe –B –K PlanSum.ctl
```

This batch file will start four instances of the Router simultaneously. However, it does not wait for them to finish before moving on, so PlanSum is also executed while the Router instances are still running. The PlanSum step fails immediately because the partitioned Plans files which are needed as input to PlanSum haven't yet been written. A potential alternative would be to use the 'start' command for first three instances only. Next consider this syntax:

```
start Router.exe –B –K Router.ctl 0
start Router.exe –B –K Router.ctl 1
start Router.exe –B –K Router.ctl 2
Router.exe –B –K Router.ctl 3

PlanSum.exe –B –K PlanSum.ctl
```

The syntax above will start four instances of the Router simultaneously. However, PlanSum in this case can only be executed without error if the final partition (partition 3) is the last of the four instances to finish. If execution of **Router.exe –B –K Router.ctl 3** finishes first then the PlanSum execution will fail since some of the plan partitions are still being calculated and written to disk. The user has no way of knowing the order in which the Router instances will finish given the random process utilized to create the HH_List files using the HHList function.

4.1 Parallelization with Python & Cygwin Scripts

The use of Windows batch files for TRANSIMS applications that are run on a single CPU is well-documented and has been utilized in a variety of test cases and model implementations. For example, the Alexandria and TestNet test case data rely on Windows batch files to control the model flow. However, the batch files are not well-suited to parallelized applications since flow control and looping would require a myriad of "if" and "goto" statements.

The favored approach to controlling parallelized TRANSIMS applications is with scripts developed in Python or Cygwin. Both Python and Cygwin software are open-source and can be downloaded at:

http://www.python.org/
http://www.cygwin.com/

Python is a general-purpose high-level programming language that emphasizes code readability and provides a large and comprehensive standard library. Python supports multiple programming paradigms, primarily but not limited to object oriented, imperative, and functional, and features a fully dynamic type system and automatic memory management. Python is often used as a scripting language, but is also used in a wide range of non-scripting contexts. Python is very well-suited for TRANSIMS applications given its flexibility and its threading module which provides automatic management of multiple simultaneous tasks.

The Python code snippet below runs four instances of the Router in parallel using Python threads. This methodology allows the process to use multiple cores on a Windows machine, in this case a quad-core machine. The code snippet creates an array of four threads and then runs them, one for each partition. When a thread finishes, a value in the 'exitmutexes' array is switched from 0 to 1. Once the process '`RunProcess`' has terminated and the array is full of 1s, the loop terminates and the script is able to move on to the PlanSun function call. This is a very simplified example for illustration purposes. This code snippet does not check the error codes associated with the Router and/or PlanSum. Therefore, if one of the Router partitions failed, the script would still initiate the PlanSum function which follows even though the Router may have terminated with errors.

```
#=================================================================
import os, thread
#=================================================================
# Run FOUR Router Instances in Parallel
def RunProcess(partition,s):
    os.system('Router.exe -K -B Router.ctl ' + str(partition))
    exitmutexes[partition] = 1

exitmutexes = [0,0,0,0]
for i in (0, 1, 2, 3):
    thread.start_new_thread(RunProcess,(i,1))
while 0 in exitmutexes: pass
#=================================================================
# Calculate link delays with PlanSum (not partitioned)
os.system('PlanSum.exe -K -B PlanSum.ctl')
#=================================================================
```

The Python code snippet below also runs four instances of the Router in parallel using Python threads. This methodology again allows the process to use multiple cores on a Windows machine. This more complex code snippet uses a defined Class '`RunProcess`' to create an array of threads and runs them, one for each partition by making a system call to execute the TRANSIMS function, in this case the Router. The second portion of the code snippet initiates a loop which joins the threads and waits for all of them to finish before moving ahead. After all threads have finished, the exit codes set by the individual calls to the operating system are read and the script exits if any of the partitions reported an error. PlanSum is again executed on a single CPU once the partitioned routing is complete, but this time the error code from the system call is also checked.

```
#=================================================================
import os, sys, threading
#=================================================================
thread = {}
exitcodes = {}
errors = 0
#=================================================================
class RunProcess(threading.Thread):
     def __init__(self,command,size,rank):
          self.command = command
          self.size = size
          self.rank = rank
          threading.Thread.__init__(self)
          self.code = 0
     def run(self):
          sys.stdout.flush()
          self.code = os.system(self.command+' '+str(self.rank))
          sys.exit()
#=================================================================
command = "Router.exe -K -B Router.ctl"
partitions = 4
#=================================================================
# Run FOUR Router Instances in Parallel
for rank in range(partitions):
     thread[rank] = RunProcess(command,partitions,rank)
     thread[rank].start()
for rank in range(partitions):
     thread[rank].join()
     code = thread[rank].code
     exitcodes[rank] = code
     if code == 1:
          errors += 1
if errors >= 1:
     sys.exit(1)
#=================================================================
# Calculate link delays with PlanSum (not partitioned)
command = "PlanSum.exe -K -B PlanSum.ctl"
code = os.system(command)
exitcodes[0] = code
if code == 1:
     errors += 1
if errors >= 1:
     sys.exit(1)
#=================================================================
```

Cygwin is a Unix-like environment and command-line interface for Microsoft Windows. Cygwin provides native integration of Windows-based applications, data, and other system resources with applications, software tools, and data of the Unix-like environment. Thus it is possible to launch Windows applications from the Cygwin environment, as well as to use Cygwin tools and applications within the Windows operating context.

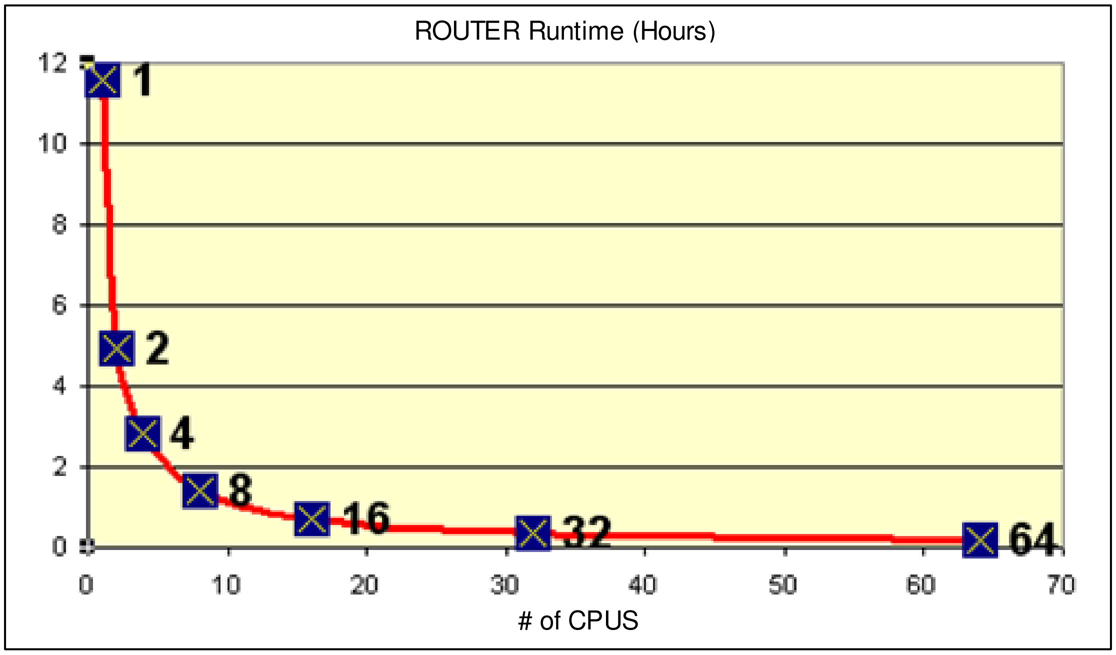
The Cygwin code snippet below illustrates the contents of a batch file that would be executed from the Cygwin shell. The batch file specifies the Router and PlanSum functions and the control files Router_All and PlanSum_Update_PrevPlans respectively. In this example, 8 partitions will be utilized for routing. The batch file calls another bash shell script called ‘runpar’ which performs some checks and writes to two files, cmd_shell.txt and cmd_batch.txt. The file cmd_shell.txt is the file that is actually executed by ‘runpar’ while cmd_batch.txt contains just the current line from the batch file for restart purposes upon errors.

```
Contents of batch file executed from Cygwin shell:
----------------------------------------------------------

./runpar Router Router_All 8 //testdirectory
./runpar PlanSum PlanSum_Update_PrevPlans 1 //testdirectory
```

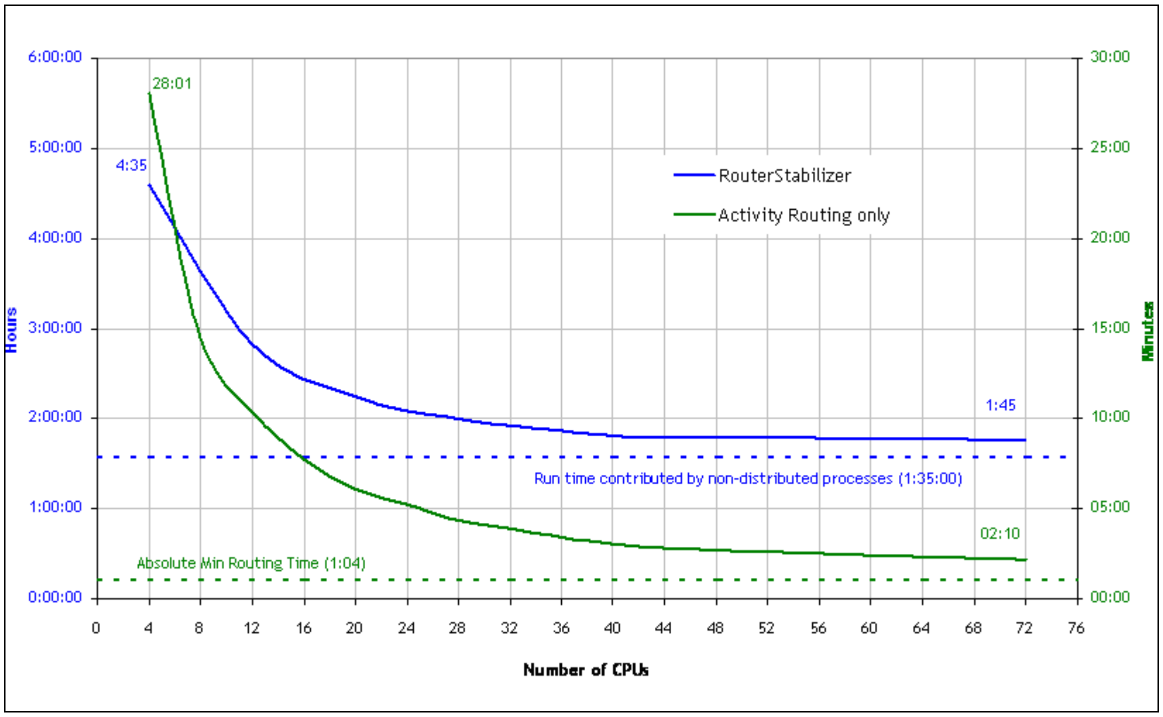
```
Excerpt from 'runpar':
-------------------------

##-- Check if running under re-try (status=2)

read prev_status < status
if [ $prev_status -eq 2 ]
then
    FAIL_EXIT_CODE=1
    echo
    echo "Currently Running $2 Under Retry Mode"
    echo
fi

##-- check if the previous run has completed successfully

read prev_status < status
if [ $prev_status -eq 1 ]
then
 echo
 echo "Previous step to $2 did not finish successfully for Run $run"
 echo
 exit 1
fi

##-- construct the command that will execute the program

command=""

curr_part=0
index=0

if [ $nofPartitions -eq 1 ]
then

    command="$command ssh ${machines[$curr_part]} ${bin}/$program -N -Q
${project}/control/${ctlFile}.ctl & "

else

    while [ $index -lt $nofPartitions ]
    do
       command="$command ssh ${machines[$curr_part]} ${bin}/$program -N -Q
${project}/control/${ctlFile}_${partition[$index]}.ctl & "
       ((curr_part = (index + 1) % num_machines))
       ((index +=1))
    done

fi

command="$command wait"

##-- now run the command

echo "`date` == $command"
echo "$command"                      > cmd_shell.txt
echo "./runpar  $1  $2  $3  $4  $5" > cmd_batch.txt
chmod +x cmd_shell.txt
chmod +x cmd_batch.txt
./cmd_shell.txt
```

5.0 Parallelization Performance Gains

As described earlier in this How-To, utilizing additional CPUs leads to a roughly linear gain in performance. That is to say, runtime to route a trip or activity list will be effectively cut in half if twice as many CPUs are utilized to perform the routing. Recent work in Chicago illustrated that routing 28 million daily trips on 1 CPU took approximately 12 hours. The figure below illustrates how the Router runtime was reduced when 2, 4, 8, 16, 32 and ultimately 64 CPUs in a cluster computing environment were used to perform the initial routing. The routing takes less than 15 minutes when 64 processors are utilized.



The most significant performance gains are achieved when moving from 1 to 16 CPUs. However, the plot above illustrates how performance gains are not as significant when the number of resources is increased significantly beyond 20 CPUs. This plot only provides a look at the runtime reduction associated with running just the router. However, as we've learned the iterative phases such as the Router Stabilization process make use of other tools in addition to the Router which can also be run in parallel. The figure below illustrates runtimes from a Router Stabilization process that was tested on a computer cluster housed at Argonne National Laboratory's Transportation and Analysis Computing Center (TRACC).



A runtime floor is approached (represented by blue dashed line) as the number of CPUs utilized in the run increases. This is due to the fact that TRANSIMS tools such as PlanSum and LinkDelay cannot be distributed and therefore contribute a fixed amount of time to each iteration that is not reduced by increasing the number of CPUs. In addition, the response of simply routing the activity list (green line) with more and more processors closely resembles the runtime reduction observed by running the full Router Stabilization process (blue line). This demonstrates that the Router is most responsive to a high degree of parallelization. However, the figure illustrates that a point of diminishing returns is observed around 20 to 24 CPUs. This finding suggests that a “mini-cluster” with just 4-5 nodes (quad-core nodes) may be sufficient to develop, maintain, and run complex dynamic traffic assignment models such as TRANSIMS.