White Paper:
Parallel Processing with HyperSizer

Phil Yarrington, Craig Collier, Mark Pickenheim
Collier Research Corporation
September 2001

Overview

The commercial HyperSizer® software performs structural analysis and sizing optimization. It can be operated interactive or in batch mode on a single workstation. A parallel capability that executes and controls the entire process over multiple computers has recently been implemented. In either use case, the process begins by identifying parts of a finite element model into manufacturing sized pieces called structural components. When sizing a structure, these components are collected together into organizational units called optimization groups. A feature of the sizing process is that groups can be executed independently, lending itself well to parallelization. This document describes the design and implementation of the HyperSizer parallelization approach.

HyperSizer was parallelized such that each group, or sets of groups, can be sent to different processors, with the ultimate goal of reducing the clock run-time of the HyperSizer analysis. The parallel version of HyperSizer has the following features:

- Only one HyperSizer parallel server license is required to execute HyperSizer on as many computers and processors as desired.
- The individual computers on the network, rather than the central controlling computer perform the majority of HyperSizer I/O and numerical processing so that for long running processes, a high parallel to serial computing ratio can be achieved.
- The network traffic is minimized between computers by allowing each workstation to perform its own disk I/O. This means the HyperSizer parallel process running over a network of computers does not bog down the network.
- Database import and management of temporary files is done automatically and is transparent to the HyperSizer user. All of the computed results are automatically imported into the database of the controlling computer.
- The HyperSizer process runs on each computer as a "low-priority" task, so that workstations running part of the HyperSizer parallel process can be used for other applications with very little impact to a workstation interactive user.
- The user can dictate the workstations on the local network on which to execute HyperSizer, can determine the relative performance of each (e.g. CPU clock speed, available memory), and tests each computer to ensure that HyperSizer is properly installed before attempting to submit a run.
- HyperSizer groups can be divided among processors on one multi-processor computer or can take advantage of processors on a network of computers. The number of processors available and the relative performance of each processor determine the division of groups.

An example running parallel on six processors

As a test problem, the HyperSizer parallel capability was applied to a candidate 3rd Gen RLV design. Using six single processor computers networked together, we were able to obtain a

speedup of 5.08 over traditional HyperSizer. The networked computers each have different processor speeds as well as differences in memory, hard drive space, and installed software. The results are shown in Figure 1. By taking into account the total processing power available on the network, a minimum processing time (if the parallelization of the problem was perfect) of 7.47 minutes was calculated. The actual time that it took to complete the problem was 8.82 minutes. By multiplying the theoretical minimum process time (7.47 minutes) by the theoretical perfect speedup factor of 6, a composite average process time of 44.8 minutes was obtained. Finally, the average time that it took each processor to complete the job as a standalone serial process was calculated to be 50.72 minutes. Additional details from this example are in the "Example Setup and Results" section below.

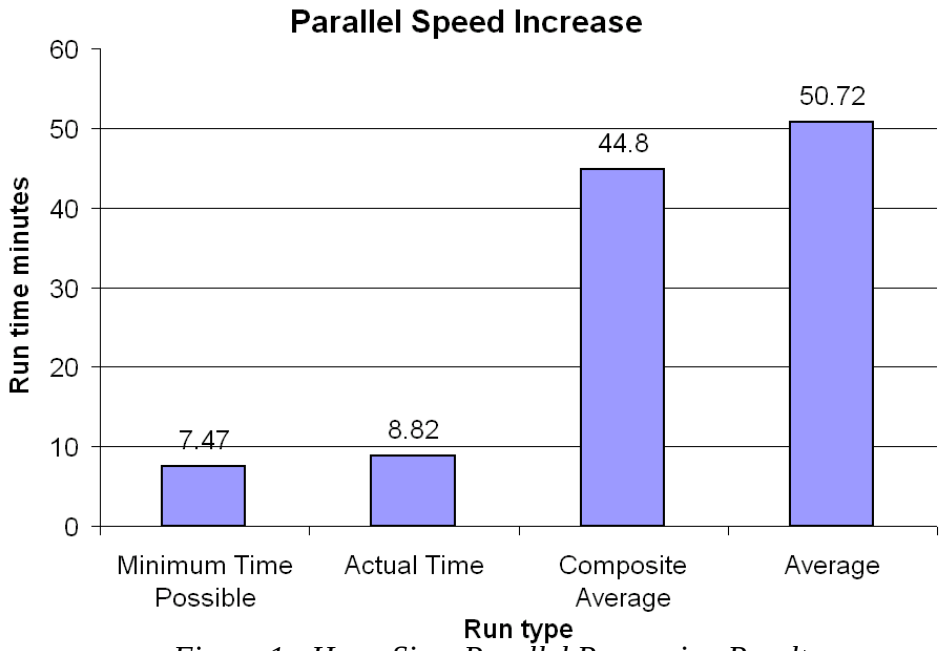


Figure 1: HyperSizer Parallel Processing Results; 6 computers used, speedup factor: 5.08

Implementation

The creation of the HyperSizer Object model has presented an opportunity for a great deal of automation of the HyperSizer structural sizing process. For example by instantiating HyperSizer objects, a user can start HyperSizer, create and size components, groups, assemblies and projects, and retrieve results. In addition, because the HyperSizer Object Model is built with Microsoft's COM and ActiveX, it can be access remotely with very little effort by the end user with the Microsoft Windows built-in remote processing facility called DCOM (Distributed Component Object Model).

The HyperSizer parallel capability uses DCOM to remotely create HyperSizer

Note: Because HyperSizer Parallel works identically for multiple networked computers and for single multi-processor computers, the terms processor, computer, workstation and machine are used interchangeably in this document. In addition, "remote" refers to any processes other than the central controlling process and could refer to a process on the same computer as the controlling process.

Object Model Processes on each networked computer, and then monitor these processes to determine when the overall sizing is complete. This could be done almost exclusively using the object model on each computer, however, some minor modifications were made to the process to prevent database conflicts where multiple computers were accessing tables within a single HyperSizer database simultaneously.

To avoid these database import and export collisions, the HyperSizer parallel process was broken into three phases.

Phase One: Initialization

In the first phase, shown in Figure 2, a HyperSizer object is created on each of the remote computers. If a PC has multiple processors, then a separate HyperSizer object is sent to each processor. After creating the object, that object is told to open a central HyperSizer database on the controlling machine. If successful, the remote object returns a "Success" flag to the central controlling computer. Because the controlling computer waits for a success or failure from each machine before continuing to the next, this initialization process is serial. If any of the processes on the remote computers fail to initialize, the parallel process is terminated and the controlling computer shuts down the HyperSizer objects on each processor.

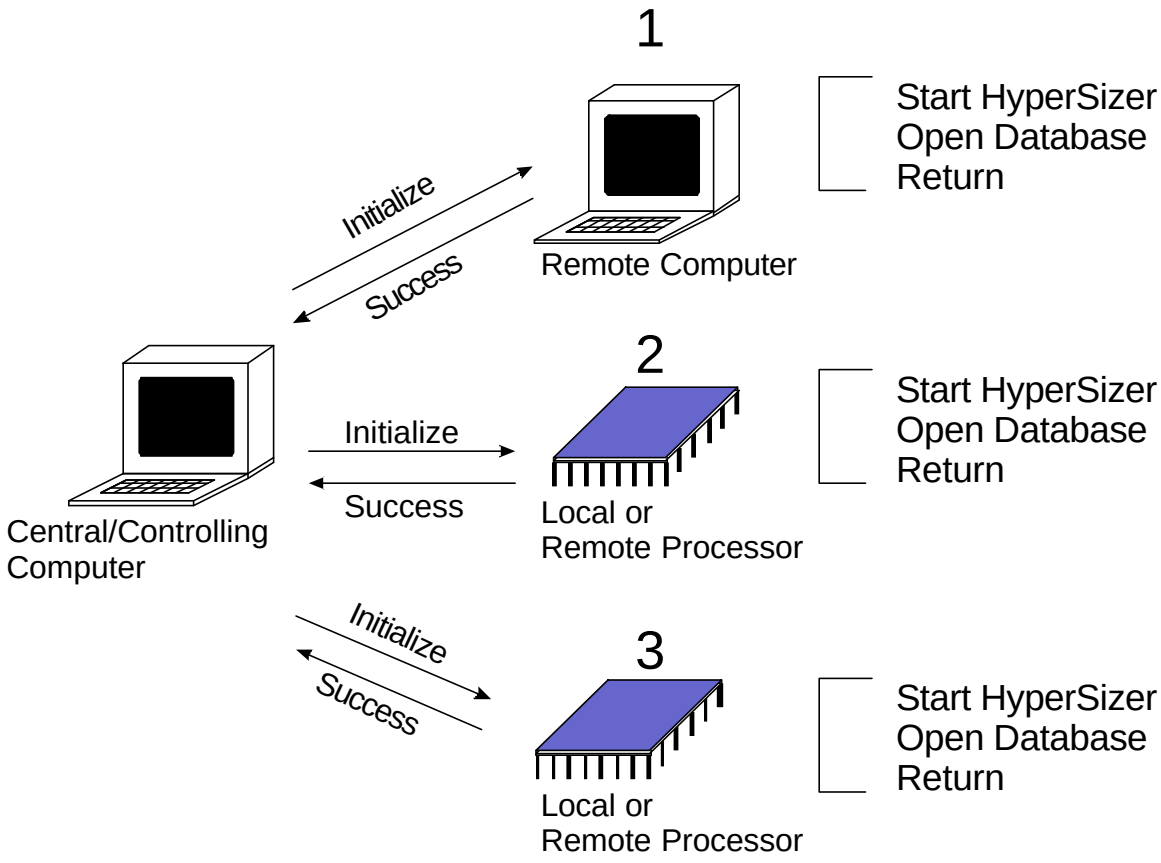


Figure 2: HyperSizer Parallel Initialization Process – A serial operation

Phase Two: Start

In the second phase, shown in Figure 3, the HyperSizer process is kicked off by the central process for each remote computer. The remote HyperSizer objects first obtain all of the sizing data for the groups that it is assigned to size from the central database on the controlling computer. After exporting this information from the database, the object spawns a process that actually performs the analysis/sizing. After successfully spawning this process, the remote computer passes a flag back to the controller to tell it that it was successfully started.

When the remote object passes its state back to the controller, the controller then goes to the next processor, however the spawned process continues to run and the remote HyperSizer object that spawned the analysis keeps track of the state of this process. At this point, the spawned processes are running in parallel. For typical sizing problems, these spawned analysis processes are by far the most CPU intensive processes in the sizing procedure. Therefore, except for the initial database export, the Start phase of the sizing is done in parallel.

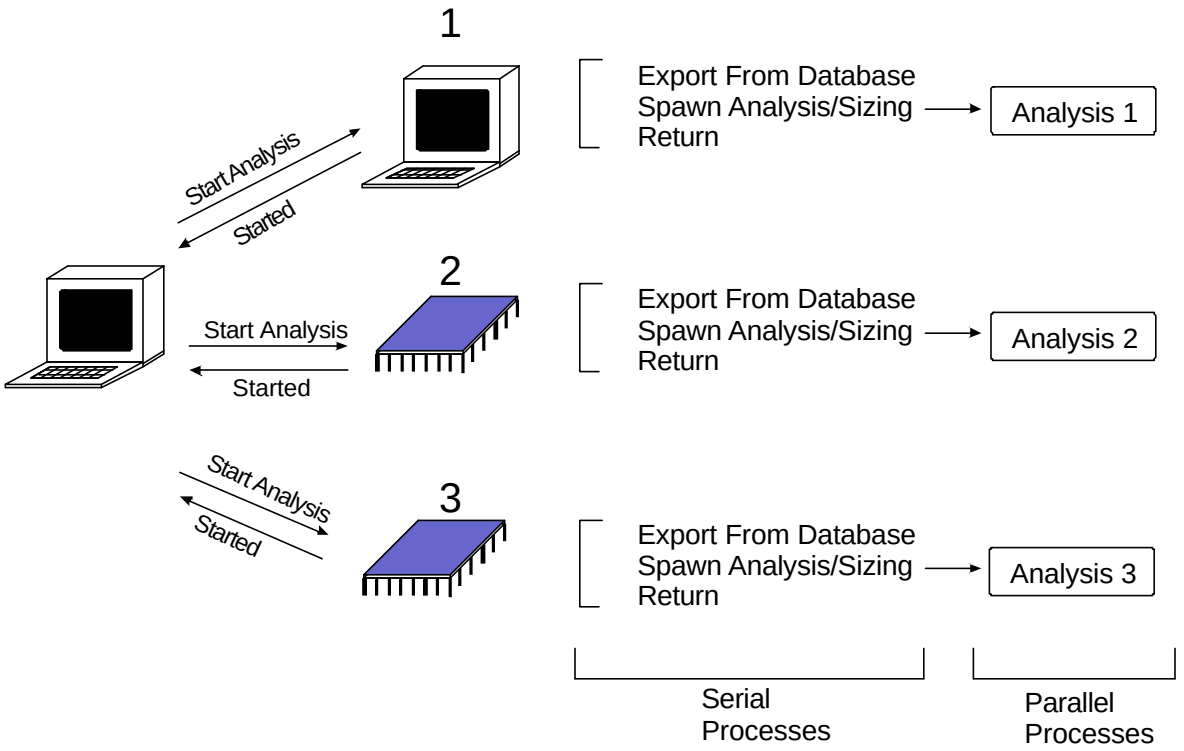


Figure 3: HyperSizer Parallel Start Process – A parallel operation (mostly)

Phase Three: Finish

The final phase, shown in Figure 4, takes place after the spawned analysis jobs have been completed. The controller keeps track of which machines still have processes running, and at regular intervals, queries each of the remote HyperSizer objects in turn to determine if their respective analyses have been completed. When an analysis is complete, the central controller instructs the remote object to import its results to the central database. During this procedure, the central controller waits until the database

import is complete before doing anything else. After completing the database import, the controller tells the remote object to shut itself down, removes it from its list of running computers, and then resumes monitoring the computers with analyses still running. This procedure continues until all remote computers have completed their analyses, imported results to the central database, and shutdown. At this point, the HyperSizer parallel sizing is complete.

Because each remote Finish process must be completed before any other remote processor can begin to Finish, this operation is serial. For example, if all of the spawned processes complete at the exact same time, then each would wait in a queue until the

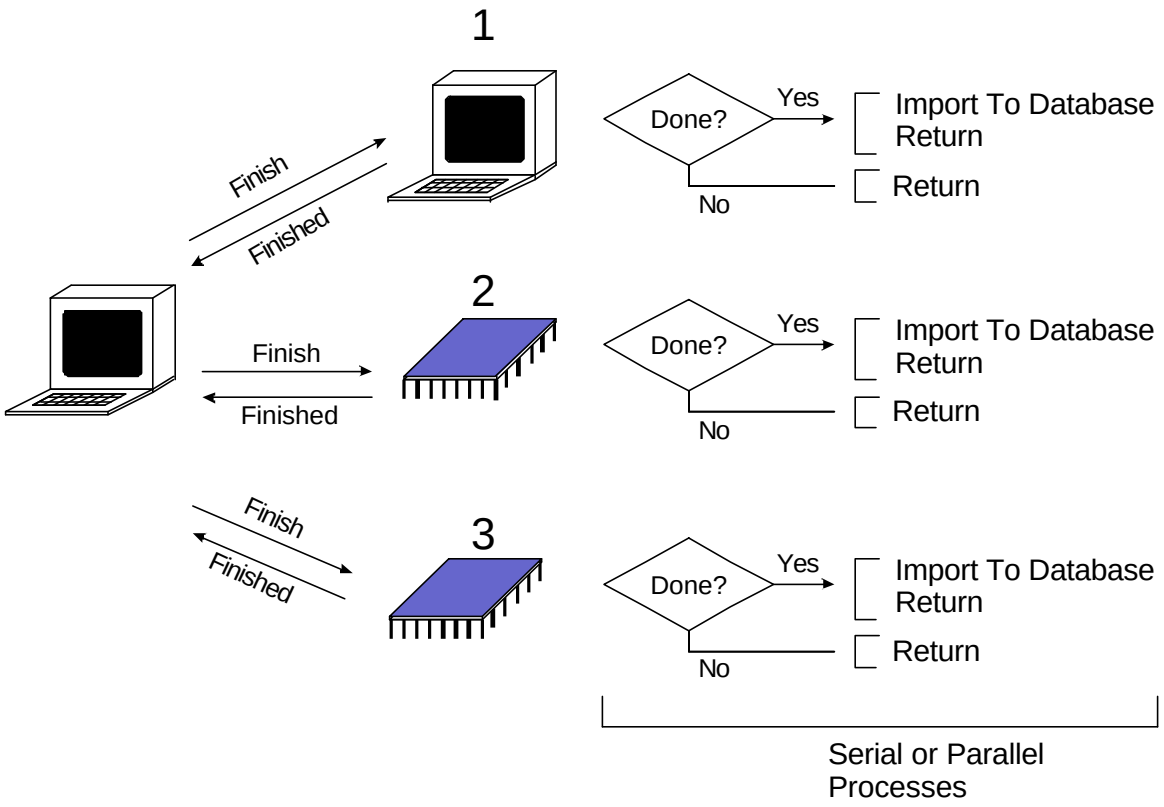


Figure 4: HyperSizer Parallel Finish Process – A serial or parallel operation

other processes were completed. In reality, however, the processes will very seldom return at the same time, so that one processor can be finishing and importing results to the database, while the other processors are still performing their analyses. The ideal setup is one in which each remote processor returns and begins its Finish procedure just as another remote computer completes its own. In this way, the Finish procedure can remain parallel in nature.

Speedups obtained using the HyperSizer parallel capability are discussed in the example problem discussed in the next section.

Example Setup and Results

As an example, the HyperSizer parallel analysis was applied to a "typical" RLV design to quantify its performance. The design is a 3rd Gen RLV concept called GTX (Shown in Figure 5), which was studied by NASA Glenn Research Center. In this example problem, we re-analyze/optimize the entire engine of the GTX vehicle, that was analyzed several months ago using the normal, serial HyperSizer software.

Figure 5: GTX 3rd Gen RLV Concept

The problem was analyzed using six computers from our office network with processor speeds ranging from 450 MHz to 1700 MHz. The first step was to run the GTX problem on each computer to gauge the relative speeds of each computer running HyperSizer. The results were all normalized against the fastest of our 1700 MHz PCs, which was used as a benchmark. The actual and normalized execution times for each computer are listed in Tables 1 and 2 respectively.

Table 1: Actual CPU times for execution of GTX optimization

PC	MHZ	Memory	Total run time in (seconds)	Total run time in (minutes:sec)	Data I/O files (seconds)
R10	1700	512	2001	33:21	7
R8 notebook	850	256	2167	36:07	10
R11	1700	512	2485	41:25	10
R7 notebook	700	192	2510	41:50	13
R6	600	256	3397	56:37	18
R9	450	256	5696	1:34:56	22
Average	***1000***	***300***	***3043***	***50:43***	***13***

Table 2: Normalized CPU times for execution of GTX optimization

PC	MHZ	Memory	Total run time (Normalized)	% CPU power of total	Data I/O files (Normalized)
R10	1700	512	1	.223	1
R8 notebook	850	256	1.08	.207	1.42
R11	1700	512	1.24	.180	1.43
R7 notebook	700	192	1.25	.179	1.86
R6	600	256	1.70	.1315	2.57
R9	450	256	2.84	.0787	3.14
Average	***1000***	***300***	***1.52***	***0.9998***	***1.90***

Theoretically, with a perfectly parallel process, the time to complete the project with all six computers running in parallel is 7 minutes and 28 seconds. This is obtained by finding the "total computing power" of the system, which is

$$1+\frac{1}{1.08}+\frac{1}{1.24}+....=4.47$$

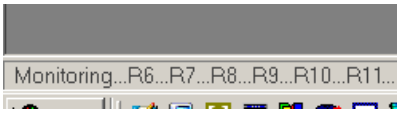
Then dividing the processing time by this number gives us the theoretical lower limit:

$$\frac{2001}{4.47}=7:28$$

The % CPU power of each computer is then calculated for each computer.

$$\text{R10: } \frac{1}{1\cdot 4.47}=.223 \qquad \text{R8: } \frac{1}{1.08\cdot 4.47}=.207 \qquad \text{R11: } \frac{1}{1.24\cdot 4.47}=.180$$

We then attempt to balance the parallel job over the network of available PC's so that R10 is processing about 22.3% of the project's groups, R11 about 18.0%, etc.



Currently, the entire time to run the project in parallel on the six computers is 8:49 minutes. A possible measure of the speed-up factor is to divide the average solo run times of all computers by the actual parallel run time.

$$S=\frac{50:43}{8:49}=5.75$$

Doing this shows that the speed increase is 5.75. However, this is idealized somewhat and the approach is flawed because this math would show a ratio greater than 6.00 when the parallel run time gets close to the limit of 7:28.

Therefore, this same equation is used to back out the composite solo run time of 44:48 for comparative speed increase purposes.

$$\frac{44:48}{7:28}=6.00$$

So the appropriate ratio is:

$$S=\frac{44:48}{8:49}=5.08$$

Finally, the ratio of actual speed increase to the theoretical increase is getting close to a one-to-one ratio:

$$\frac{5.08}{6} = 85\%$$

Amdahl's Law

Amdahl's law is difficult to apply to this problem because in its truest form, it applies to parallel processing where all processors are exactly the same. In our case, every computer has a different processing speed, different memory, hard drive space, etc. By using the speed up factors calculated above however, we can attempt to back out the performance by Amdahl's Law, and then use it to predict future behavior. Amdahl's Law is given by:

$$S = \frac{1}{(1-F)+\frac{F}{P}}$$

where *S* is the speedup factor, *F* is the fraction of the sequential code that can be parallelized, and *P* is the number of processors available for parallel operations. We solve this equation for *F* to obtain:

$$F = \frac{S-1}{S\left(1-\frac{1}{P}\right)}$$

In the idealized case where the speedup factor, *S*, was calculated to be 5.75, the fraction of parallelizable code, F, comes out to be

$$F = \frac{5.75-1}{5.75\left(1-\frac{1}{6}\right)} = 0.991$$

And in the more realistic case, *S* = 5.08, the ratio comes out to be

$$F = \frac{5.08-1}{5.08\left(1-\frac{1}{6}\right)} = 0.964$$

Plugging these two ratio's into Amdahl's law, we can predict the behavior of the parallel system for an increasing number of processors.

$F = 0.991$		$F = 0.964$	
P	*S*	*P*	*S*
2	1.98	2	1.93
10	9.25	10	7.55
100	52.88	100	21.9
Limit	*111.1*	*Limit*	*27.8*

The actual parallelizable ratio, F, probably lies somewhere between these extremes, but these results do show the limitations of the parallel method. We can conclude that if we were to apply 100 processors to this problem, that we would expect more than 22 and no more than 52 times speedup.