SESUG Paper 111-2023

Speeding up Your SAS® Code Using Parallel Processing

David L. Ward, InterNext, LLC

ABSTRACT

This paper will present a history and overview of parallel processing, examine the types of operations that make best use of parallel processing, catalog various techniques and products available within SAS®, and present real-world examples and strategies for doing your own parallel processing. While network architectures such as virtual machines and containers along with SAS products such as SAS/Viya® and SAS Grid Manager will be explained, the code examples in this paper will all require only Base SAS and will be aimed at performance increases on a single, multi-core machine. Depending on the types of operations you intend to parallelize, you could see more than a 90% improvement in clock time. While SAS has implemented multi-threading in many of its procedures for a long time, this paper will explain the difference between multi-threading and multi-processing and provide practical code that is understandable and extensible even to novice SAS programmers.

INTRODUCTION

Parallel processing refers to the ability to process multiple tasks simultaneously. Since the earliest days of computing, most programs and programming languages have been designed serially - to process one instruction at a time - in a logical, usually top-down sequence. So it is with SAS programs: our code executes one line after another with just a few exceptions like data step and macro goto statements.

Most of our SAS programs are designed to manipulate data in a sequence where each step is not always dependent on the steps that come before it. Consider the need to read file A into a data set, sort it, then read file B into a data set, sort it, and then merge them. In this case, reading and sorting files A and B could be done separately, and the merge step would need to wait until they are both finished; alas, the SAS language does not give us that capability within the language.

The reason that you should be interested in thinking about parallel execution is that you can often massively speed up your programs. Modern computing architectures have many available processors, incredible amounts of fast memory, fast networks, and fast CPUs. Still, our programs spend a lot of time waiting and underutilizing most of these resources. Because advances in hardware continue at an incredible pace, designing your programs to take advantage of parallelization can reap exponentially increasing benefits without significant changes to the code.

OVERVIEW

This paper will guide you through the following sections:

- What is parallel processing: an overview, history, and defense.
- Key operational constraints: various bottlenecks that parallel processing purports to work around.
- Hardware and network architecture: processors, networking, virtualization, and in-database technologies
- SAS products and their history: How SAS has solved the parallel processing problem over the years and what products are offered.
- Base SAS options: A variety of options and techniques within base SAS that most readers will have immediate access to.
- Putting it all together: Example problems and how to approach re-architecture.

Key takeaways

- A basic conceptual understanding of parallel processing.
- The ability to assess whether and how parallelism might be appropriate for particular programs/applications.
- An understanding of various SAS products that offer parallel processing capabilities.

- Some Base/SAS techniques that provide immediate usefulness no matter what SAS products or platforms are available.

WHAT IS PARALLEL PROCESSING?

As stated above, parallel processing is the ability to process or run multiple tasks simultaneously. While this might seem simple, the evolution of computer hardware has produced a string of terms that are sometimes used interchangeably but have different meanings. This orientation should prove helpful for a proper understanding of the remainder of this paper and in future reading or discussions as you seek to implement this in your code.

THREADING/MULTI-THREADING

Threads in computer science refer to a unit of "work" that can be paused and started again quickly. Pausing and starting again is essential when threads are used in non-parallel or serial processing. If reading a book were a program, you might consider each chapter a thread. If you were reading a mystery novel, you would want to read each chapter in sequence and might place a bookmark at the chapter where you finish off each time you read. If you were reading a cookbook, the order of chapters isn't essential.

Threads can be used to describe units of work that are either serial or parallel. Imagine that you and your classmates procrastinated to avoid reading a 1,000-page sociology textbook until a week before the final exam. To solve your dilemma, you assign ten chapters to 10 classmates and decide to gather a day before the exam and have each person give a 10-minute presentation condensing the material and highlighting the most critical information. In this example, you and your friends have utilized the parallel execution of threads to finish reading the book (the whole program) in less time.

When threading is used serially, it refers to context switching or the ability of the processor to switch back and forth between various tasks without waiting for one to finish. This is how humans multi-task (at least for mental tasks). I have heard it said that no human can *truly* parallel process but that instead, some are good at context switching that *appears* parallel.

SAS uses the term threading heavily to describe how it added parallel processing to base SAS. See their "Threading in Base SAS" documentation page for more information. While the terms multi-threading and multi-processing (or parallel processing) are not always synonymous, this is how SAS has used them. This is likely because there is little use for the types of problems that serial multi-threading can solve (particularly network servers), making the concept simpler to understand.

A BRIEF HISTORY OF PARALLELIZATION

CPUs

True parallelization occurs when multiple threads or tasks can be executed simultaneously on different *processors*, so none of them needs to wait until another one finishes before starting or continuing. This became possible early in computing with the advent of shared memory processors. When multiple processors were connected that could access the same memory (so that instructions and results could be returned to a calling program), parallel processing became possible.

From "simple" shared memory processors came mass-market processors or MPPs, which brought the ability to string together cheaper and readily available processors onto a shared memory system. More recently, the advent of CPUs with multiple "cores" has made parallel processing affordable and easily accessible to the home PC market. Operating systems like Windows make heavy use of multi-processing seamlessly and often invisibly. The subsequent advances in processing involved networking as an extension of the shared memory system.

Networking

When computer networks became available, ways were devised to connect clusters of disparate systems together so that processing could be distributed. When the Internet became available, this technique has and continues to make parallel processing available – particularly for scientific research when users

agree to allow their computers to be used in this way – at a massive scale. While the sharing of information across a network like the Internet is orders of magnitude slower than memory sharing, the types of data-intensive problems solved in this way do not need large amounts of information updated frequently. This is the same technology that some SAS products like SAS Grid Manager utilize.

Disk

Disk technologies have also advanced the ability of computers to send/receive, or read/write data in parallel. From ATA to SCSI, the ability of disks to break apart a bitstream into pieces (not customarily called threads, but the paradigm fits) has made I/O (input/output) faster with each new technology. RAID gave us the ability to connect multiple drives. Listing the dizzying array of technologies available today that speed up modern disks is outside of the scope of this paper; however, mentioning these advances is meant to reinforce the point that *our disks can usually handle much more data at a faster than our (serial) programs typically throw at them*.

KEY OPERATIONAL CONSTRAINTS (BOTTLENECKS)

We can think of the computing problems parallel processing attempts to solve as operational constraints or bottlenecks. It is vital to delineate the types of constraints our programs face so that we use the right tool for the right job. This will help us avoid a laborious re-architecture- only to discover that it did not have the intended effect because we cut off the wrong bottle tops! For example, if your program is downloading ten files from an FTP server and you re-write it so that those downloads happen in parallel, you may not get much or any improvement in speed because of the bandwidth limitations. If you have a 100Mb connection, you can download one file at 100Mb or ten files at 10Mb each, which will take roughly the same time.

CPU

Central processing units, or processors, are the natural and first constraint to consider. After all, “processor” is built into the term “multi-processing!” Since the processor handles the instructions, it is the first place where parallelization can usually have a dramatic effect. We call a program “CPU-intensive” if it heavily uses the processor. If you have a complex data step or analytic procedure that is doing a lot of operations on data and manipulating it in memory, this uses a lot of CPU. Contrast this with a program that reads a giant flat data file into a SAS data set using minimal CPU. An easy way to get a feel for how CPU-intensive your program may be is to watch the system monitor (Windows resource monitor or Linux top command) as your program runs.

Memory

The next operational constraint to consider is memory. Both the speed and capacity of memory will affect performance. When modern operating systems are asked to provide more memory than they physically have, they will start using disk as virtual memory (often called the page file). As you can imagine, using a disk instead of physical memory will have a massive effect on your application, and it may not be readily apparent what is happening. While your program is running, again, look at the resource monitor to see how much memory it is consuming and how much the whole system is using. The SAS setting MEMSIZE has a default setting of 2GB (per session) to prevent SAS from “taking over” the machine by utilizing too much memory. This invocation option can be changed since many modern systems have far more memory available. Memory-intensive programs involve operations like table lookups (think hashing, proc format, or proc SQL joins) or heavy analytic procedures.

Network

If your program reads or writes data across a network, this can become a bottleneck. The example above of downloading files from an FTP server perfectly illustrates this. Think of a network connection as a pipe; only so much liquid can flow through the pipe. Resource monitor tools will show you how much bandwidth your program uses. You can use this to identify the data transfer speed and compare it with the expected throughput capacity. Network speed is notoriously tricky to measure. For example, if you have a Gigabit network card and local network infrastructure (cables, switches, etc.), you will typically not

see speeds very close to that on consumer computers. The hard drives we use to save the data often cannot keep up, so your bottleneck will be a combination of network and disk.

Remote servers often have a bandwidth limit set on streams to individual connections. Because of this, it is possible to exceed this limit by using multiple concurrent connections. The server software or configuration is typically unavailable, so some trial and error is needed. Many modern FTP clients allow several concurrent connections (multi-processing!). Observing the speed of each transfer and changing the number of connections will reveal whether it might be possible to speed up your download step by breaking it into parallel processes. Sometimes, network bottlenecks are impossible to identify because of the invisible and untamable complexity of the network path between your program and the remote server.

Disk

I/O is also a common bottleneck in SAS programs. CPUs can process data much faster than disks can provide (input) or receive (output) it, so the processor is doing a lot of waiting and will be visibly underutilized. Care should be taken to understand the types of disks available on your system(s), using the fastest disks for the most I/O intensive parts of your programs, copying data to more permanent disks when finished.

Most modern disks have built-in parallel processing capabilities (like SSDs) or can organize and prioritize read requests from multiple cores to more efficiently handle concurrent requests (HDD). If you have ten files to write to disk, you can almost always write those files to disk faster using parallelization. Again, trial and error should be used along with research into your hard drive specifications to determine how much parallelization to use. At some point, too much parallelization will start degrading performance as the drive's capacity is reached.

Black Box

Another type of operational constraint should be named: we shall call it the “black box.” A black box refers to a computing resource (hardware or software) that is a closed system, often provided by a commercial company that sells and supports it. Database appliances could be called black boxes because while some documentation might be available and some configuration might be possible, there will always be a hard limit to your understanding and control of its behavior. Black box software could be any application that your program calls for which you do not have source code. This is perhaps the most frustrating bottleneck because there is often nothing that can be done about it!

HARDWARE AND NETWORK ARCHITECTURE

Understanding how your machine and network are designed and the various virtualization techniques available will further expand your parallelization knowledge and strategies.

WITHIN ONE MACHINE

Cores refer to individual components within one CPU chip that provide parallel processing, appearing as separate CPUs to the operating system. Various manufacturers have gone even further with their architecture to add what is typically called “hyper-threading,” a technique that allows simultaneous multi-threading even within one core, appearing as virtual CPUs to the operating system. For example, the machine that I am using to write this paper has a chip made by Intel with the following specs:

Processor Number	i7-1185G7E
Total Cores	4
Total Threads	8
Max Memory Size (dependent on memory type)	64 GB
Memory Types	DDR4-3200, LPDDR4x-4267

Display 1. Specifications of an Intel CPU

Windows task manager reports that four cores and 8 “logical processors” are available; the Resource monitor shows eight independent graphs of how heavily the processors are being utilized. While researching how to speed up your programs, consider upgrading your CPU or your system’s memory. These specs list a max memory size of 64GB. The type of memory can also impact how fast the processor can handle threads.

With the advent of machine learning and artificial intelligence, GPUs have gained much more interest. Graphics Processing Units are designed intrinsically for parallel processing. While they are best known for use in gaming and graphics (hence the name), they are being leveraged more by software and programming languages for massively parallel operations. SAS mentions utilizing GPUs in SAS/Viya and some specialized tools, but it is not widely available within the SAS system yet, thus outside of the scope of this paper. But if you’d like to understand the hardware behind some of the most cutting-edge areas of computing today, do some outside research on GPUs!

Memory Sharing

A requirement for parallel processing to function is the ability of computer systems to share memory across processors. This allows threads running on one processor to start up threads on another and pass information back and forth. Many applications and languages are designed to be “memory safe,” ensuring that the memory addresses used or reserved for one thread are not unexpectedly changed by another. Or vice versa, your application cannot accidentally (or intentionally, in the case of malicious code) write to memory used by other threads, causing unpredictable results or operating system crashes. While SAS does provide functions like peek() and poke() that can access memory, this is a very advanced feature that most doing data-processing work will not encounter.

When utilizing threaded procedures provided by SAS, you usually do not need to consider how the threads communicate; SAS takes care of that in the (presumably) most efficient manner (whether memory or disk). When doing your own parallel processing using the base SAS, you must consider how you will pass instructions and data into and out of your threads. Invocation options such as sysparm, environment variables, or pipes can be used. Since STDOUT is unavailable to SAS on Windows, output files or environment variables can be used to get information back from threads.

DISTRIBUTED (NETWORK) PROCESSING

The network that your programs are running on may yield previously unknown parallel processing capabilities. While SAS products installed on a network like SAS Grid Manager leverage multiple computers “easily,” SAS code can manually be broken into multiple threads and executed via remote commands such as SSH in Linux and PowerShell in Windows. Then, assets or results can be retrieved back to the master program for tabulation/finalization. Some complexities that will add challenges to this technique are authentication/permissions, remote resource management, and network security/firewalls

and bandwidth. Network administrators should be brought in to advise on an architecture that will not hurt other processes or the budget (especially with cloud services where you pay for usage).

Distributed network processing led to three other advances in network design worth mentioning: virtual machines, containers, and serverless (all key pieces of what is often simply referred to as the "cloud").

A virtual machine is a software abstraction layer that provides the necessary interfaces for an operating system to run within a host's system (hardware and software). Examples are VMware and VirtualBox. While this technology greatly impacted the testing and development environment/process, it does little to provide faster or more efficient processing. Each virtual machine is limited by the amount of resources defined for it by the host computer, which itself has limited resources, constraining the number of VMs that can be started.

Containers

While virtual machines are not directly useful for parallel processing, they were brought up here to help understand the next evolution in virtualization: containers. Containers are ways of "packaging" up a piece of software (or code) and all that is needed to execute it, given certain operating system assumptions. When the host computer is asked to run a container, it creates a separate virtualized yet vastly smaller environment for it, then executes whatever instructions the container comes bundled with until it is finished. Typically, containers handle smaller or shorter actions, but this is not endemic to their nature. Containers provide a quick way to scale up many parallel actions using the same code but with different configurations (think different sets of data). The prevailing container technology is Docker, with container management systems like Kubernetes providing the ability to orchestrate "swarms."

While SAS does not provide out-of-the-box solutions for use with containers/Docker, they provide some documentation on building your own container. See the article "SAS 9.4 and Container Technology: Build and Run a Container" for more information. This is a worthwhile area to research if your organization has already made containerization available.

Serverless

The serverless network architecture is the next evolution of cloud/virtual computing after containers. Containers must be run on host servers that must be maintained and configured. To support a high degree of parallelization, a host server with a lot of resources must be maintained and always available, which will typically be expensive and wasteful if those resources sit idle a lot of the time. This is where serverless comes in – cloud vendors will manage the container host server for you and only charge for usage. These host computers can be placed on giant machines with lots of resources (typically more than an organization could afford). Cloud vendors will even allow the execution of containers or container-driven code on different servers depending on the resource requirements. A container isn't even needed for some serverless technologies – just code. You can design functions in products like AWS Lambda with various triggers to execute them and do not have to manage the server it is run on. These technologies are only beginning to be explored by the SAS community since more emphasis has been placed on SAS's cloud technology, SAS/Viya. Still, there are many exciting possibilities for the eager programmer to explore.

IN-DATABASE PROCESSING

The last hardware and network architecture consideration often brings the two together. Many of our SAS programs utilize database systems such as Snowflake, Exasol, Teradata, MySQL, Redshift, Oracle, etc. Often, these database systems have a high degree of performance optimization built in that is accessible via configuration or code statements. Black box appliances systems like Netezza are called Massively Parallel Processing products for good reason – they have been designed from the ground up physically and virtually to make operations on data incredibly fast through parallelization in a way that is understandable and accessible to mortal programmers, often just using the same SQL we've known and used for years. If your organization uses a database for any part of your programs, its techniques for fast processing should be explored and leveraged. A bit more on this will be presented later under SAS/Access.

SAS PRODUCTS AND THEIR HISTORY

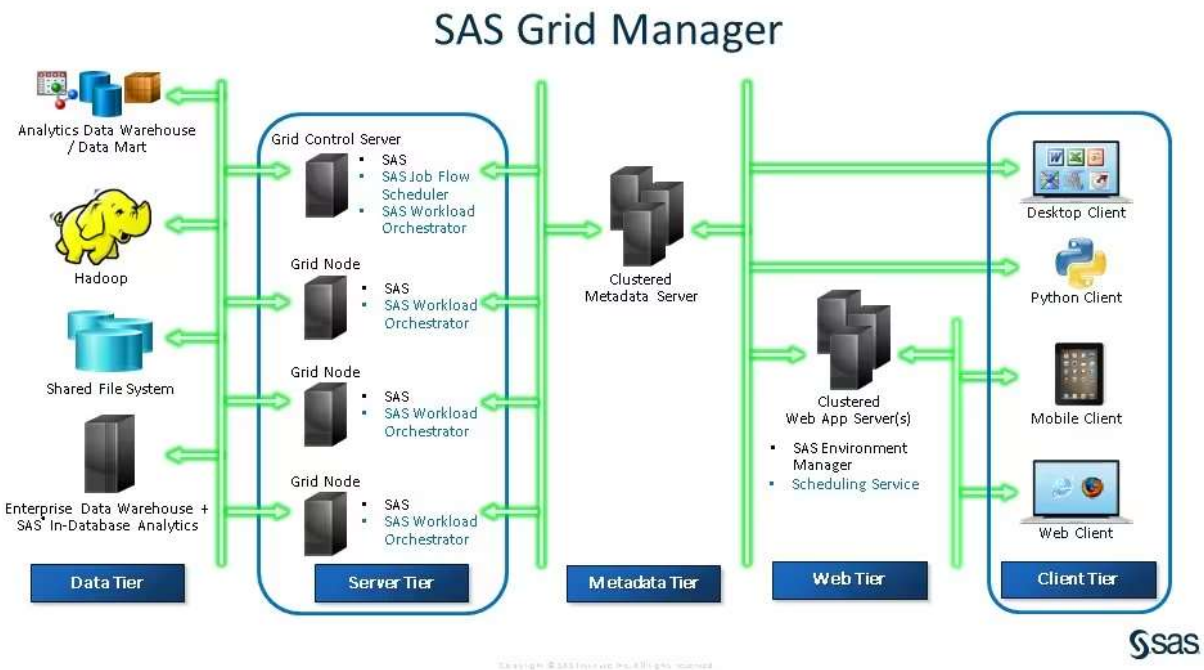
Before we get to the main practical focus of this paper – parallel options built into Base SAS – let's briefly survey some of the SAS products designed to provide scalable/parallel processing of large amounts of data. If you have these products available to you, by all means, leverage them!

SAS/Connect® and MP Connect

SAS/Connect is one of the oldest products with built-in parallelization via the WAIT=NO option on the RSUBMIT statement. The purpose of SAS/Connect is to allow one SAS session to connect to a remote (and more powerful) server on which another SAS session is invoked, run code, and then receive logs. If WAIT=NO is specified, multiple programs can be executed simultaneously, with a WAITFOR statement afterward to wait until all of the remote tasks are completed. See the helpful blog post " Running SAS programs in parallel using SAS/CONNECT" for more details. MP ("Multi-process") Connect adds a convenience layer on top of SAS/Connect to perform parallel processing with more options and more straightforward syntax.

SAS Grid Manager

The following product evolution for SAS was SAS Grid Manager, which has far more integration options than SAS/Connect for the remote submission and management of SAS code. The Grid system provides five tiers: data, server, metadata, web, and client. Not only can a SAS session initiate a remote connection, but command line tools such as SASGSUB and mobile, web, or Python clients can submit and manage jobs. The server layer is scalable and can add more nodes on demand (as long as there is availability). The data tier is designed to connect to many different types of data storage, including the file system, Hadoop, or other SAS products. But the added capability brings added complexity, so most often, you will need network administrators to work with SAS programmers to install and maintain a SAS Grid environment. The following image is from the SAS/Grid documentation and provides a helpful overview of its architecture:



Display 2. SAS Grid Manager architectural diagram

SAS In-Memory Analytics®

This product is a bit more specialized because it is designed with data processing and analytics in mind. In-Memory Analytics combines the various client/server tools SAS had written with beefed-up analytics procedures and the ability to natively connect and submit processing on specially designed database appliances such as Teradata or Greenplum. It also utilizes the Hadoop Distributed File system. SAS Visual Analytics sits on top of this server to provide a business intelligence application that can access and report on huge amounts of data using these techniques. This product will interest you if you already have experience with or license products like Teradata.

SAS/Viya

SAS/Viya represents SAS's strategy for making its software cloud-based. A key component of this product is the "Cloud Analytic Server," which is similar to SAS In-Memory Analytics but with a notable difference being much more accessibility to outside packages via interfaces such as a REST API. SAS/Viya can be accessed via the "old" familiar syntax of SAS/Connect and through new and exciting methods and languages like Python. Moreover, because the entire SAS System runs in SAS/Viya (in contrast to specific procedures with In-Memory Analytics), this opens up much more potential for highly customized parallelization, which can leverage the entire SAS/Viya server infrastructure. While more details on how to accomplish this are outside of the scope of this paper, as this is a current and heavily emphasized SAS product, there is an incredible wealth of resources available, likely at this very conference.

BASE SAS OPTIONS

Multi-Threaded Procedures

As a reminder, when SAS refers to threading or threaded execution, they mean parallel processing (though multi-threading is not always parallel). See the article "What Is Threading Technology in SAS?" for detailed information about how SAS uses both I/O and CPUs in their multi-threading. In Base SAS, they have built threading into a variety of procedures, and this is enabled by default. The following procedures leverage multi-threading via the CPUCOUNT system option (which controls how many processors are available to SAS). Since the default value is 4, you should experiment with increasing this if your system has more than four processors.

- MEANS
- REPORT
- SORT
- SUMMARY
- TABULATE
- SQL (direct, not pass-through)
- (a variety of SAS/Stat procedures)

Because these procedures have threading enabled by default, you may be using parallel processing in your SAS programs without even knowing it.

SAS/Access®

SAS/Access is technically a separate product but is usually bundled with Base SAS and is widely available. SAS/Access allows easy access to the aforementioned in-database technologies. Much of this parallelization is implicit but can be controlled with options such as DBSLICE, DBSLICEPARM, THREADS|NOTHREADS. Threading is accomplished either within the database itself or with multiple network connections that stream subsets of the data simultaneously.

SPDE/SPDS

The SAS Scalable Performance Data Engine is built into Base SAS and has been around for a long time. The server (the "S" in SPDS) is a standalone product allowing you to share and scale the processing SPDE provides on a single machine. SPDE is available as a libname engine which is "optimized for the storage and sequential access of large and huge data sets." This technique will allow you to add parallel processing of I/O with very little code required. It stores data sets in a very different format than the

typical *.sas7bdat you may be used to. In addition to this brief sample code to whet your appetite, see the paper “An Annotated Guide: The New 9.1, Free & Fast SPDE Data Engine” for an excellent overview.

While this example did not yield any performance benefits on a simple consumer laptop, the documentation explains that the typical target for large performance increases is with RAID drives:

```
Libname SPEtest
  SPDE "C:\Temp\SPDE"
  datapath=("C:\Temp\SPDE\A" "C:\Temp\SPDE\B" )
  indexpath=("C:\Temp\SPDE\C")
  partsize=64;
* SPDE writing 100M rows ;
data SPEtest.one(drop=i index=(x));
  do i = 1 to 100000000;
    x = ranuni(0);
    output;
  end;
run;
* V9 writing the same 100M rows ;
data one(drop=i index=(x));
  do i = 1 to 100000000;
    x = ranuni(0);
    output;
  end;
run;
```

FedSQL

Proc FedSQL is included with Base SAS and provides some performance enhancements over standard Proc SQL. The parallelization it offers is different enough from the threaded procedures that it warrants its own treatment. This procedure was designed to implicitly pass SQL statements through to database servers like explicit pass-through via the connect statement in proc SQL. This pushes processing into the database, which generally has parallelization built-in, and decreases the amount of data traveling back and forth between SAS and the server during step-wise ETL processing. In addition, it takes advantage of in-database optimizations such as data partitioning and special join techniques to make code run potentially much faster than proc SQL with minimal code changes. Sample code is not needed because you are encouraged to take any Proc SQL code and experiment with simply adding “Fed” to the beginning! It even works with SAS data sets, not just SAS/Access data sources.

Proc DS2

Proc DS2 (Data Step Version 2) is a reinvention of the data step designed with several modern enhancements beyond the data step, including support for more data types, access to various packages including matrix processing, and (most importantly for this paper) multi-threading.

Sample code showing performing calculations across rows on a very large data set:

```
data orders;
  do i = 1 to 1000000000;
      order_id = round(10000000* ranuni(0));
          amount1 = round(100 * ranuni(0), 0.01);
          amount2 = round(100 * ranuni(0), 0.01);
          output;
    end;
    drop i;
run;
* Use DS2 to create the sums in 8 threads ;
proc ds2;
  thread order_sum;
```

```
    dcl double total;
    method run();
      set orders;
      total = amount1 + amount2;
    end;
  endthread;
  data order_totals;
    dcl thread order_sum t;
    method run();
      set from t threads=8;
    end;
  enddata;
run;
quit;
NOTE: PROCEDURE DS2 used (Total process time):
      real time           1:49.66
      cpu time            50.21 seconds

* Use data step to do the same thing ;
data order_totals2;
  set orders;
    total = amount1 + amount2;
run;
NOTE: DATA statement used (Total process time):
      real time           2:37.13
      cpu time            8.17 seconds
```

DS2 can also utilize the in-database acceleration features you have already read about in this paper when interacting with remote databases, which have unique features that enable parallel processing. Currently, DS2 does not support I/O from files but only SAS data sets/ engine files and SAS/Access engines. This means that it is not yet suitable to work around I/O bottlenecks in reading or writing files (that are not SAS data sets).

SYSTASK

The SAS language provides the ability to execute arbitrary operation system commands via the SYSTASK statement. The system() function and the X statement also provide similar functionality but are less flexible than SYSTASK, which provides a built-in mechanism for asynchronous background processing and automatic waiting. While this last approach may be the most complex, it offers the most significant potential for parallel benefits because many separate SAS steps can be run in parallel processes. Thus, the various types of bottlenecks can be worked around with precision. The example sum from DS2 above will be rewritten as a parallel program using SYSTASK. The general approach is to use a child program (sum.sas), which will utilize &sysparm to determine which portion of a data set (using OBS and FIRSTOBS) to process and save. The calling program will queue up 10 of these and then wait for them all to finish. A final step is to assemble all ten pieces using a view instead of a data step. If more code were to follow that references this view, the results would be the same, but the time it takes to append all of the pieces together would be saved.

```
-- program sum.sas --
libname w '<data-dir>';
%let suffix = %scan(&sysparm, 1, @);
%let firstobs = %scan(&sysparm, 2, @);
%let obs = %scan(&sysparm, 3, @);
data w.order_totals&suffix;
  set w.orders(firstobs=&firstobs obs=&obs);
    total = amount1 + amount2;
run;
```

```
* 1,000,000,000 rows ;
%macro doit;
  %do i = 1 %to 10;
      %let firstobs=%sysevalf(100000000*(&i-1) + 1);
          %let obs=%sysevalf(100000000*&i);
    systask command """%sysget(sasroot)\sas.exe"" -sysin sum.sas
      -log sum&i..log -sysparm ""&i@&firstobs@&obs"" -nostatuswin
      -noterminal -nosplash" nowait;
    %end;
%mend;
%let _start=%sysfunc(datetime());
%doit;
waitfor _all_;
data order_totals_all / view=order_totals_all;
  set order_totals1-order_totals10;
run;
%put TIME: %sysevalf(%sysfunc(datetime())-&_start);
TIME: 88.6660001277923
```

This example only took 89s instead of 157s using a single data step. SYSTASK can also be used to write output files. Instead of calling sum.sas, consider calling this program (write.sas), which will write a portion of the large file into CSV files. It would be called in the same way that sum.sas was called above. This would generate 10 CSV files simultaneously, which is much faster than generating one large file. If your process can deal with the ten separate files (perhaps putting them back together at the end or just uploading ten separate files to a remote server), this approach will execute much more quickly.

```
-- program write.sas --
data _null_;
  set w.orders(firstobs=&firstobs obs=&obs);
  file "c:\temp\data&suffix..csv" dsd dlm=",";
  put _all_;
run;
```

This approach using SYSTASK is more hands-on as there are no convenience features such as automatic log or output inclusion, the sharing of parent session assets such as macro variables, macros, formats, or work data sets (like with the DOSUBL function when it executes code in a “side” session). However, these limitations can be worked around with persistence to build a more flexible and robust application.

Warning

A word of warning is essential when introducing a concept as powerful as using SYSTASK with macro looping. All computer systems have limited resources, hence the section on constraints or bottlenecks above. This is especially true on a single developer workstation, where code like this is likely to run. Care should be taken to determine appropriate values for the MEMSIZE option, which can be added to the SAS invocation command. Additionally, the number of concurrently executing processes should be limited by adding WAITFOR statements periodically. For example, the above example started ten child SAS processes; starting 100 would not be advisable as it would severely tax your local system, nullifying any performance gains you may have been attempting.

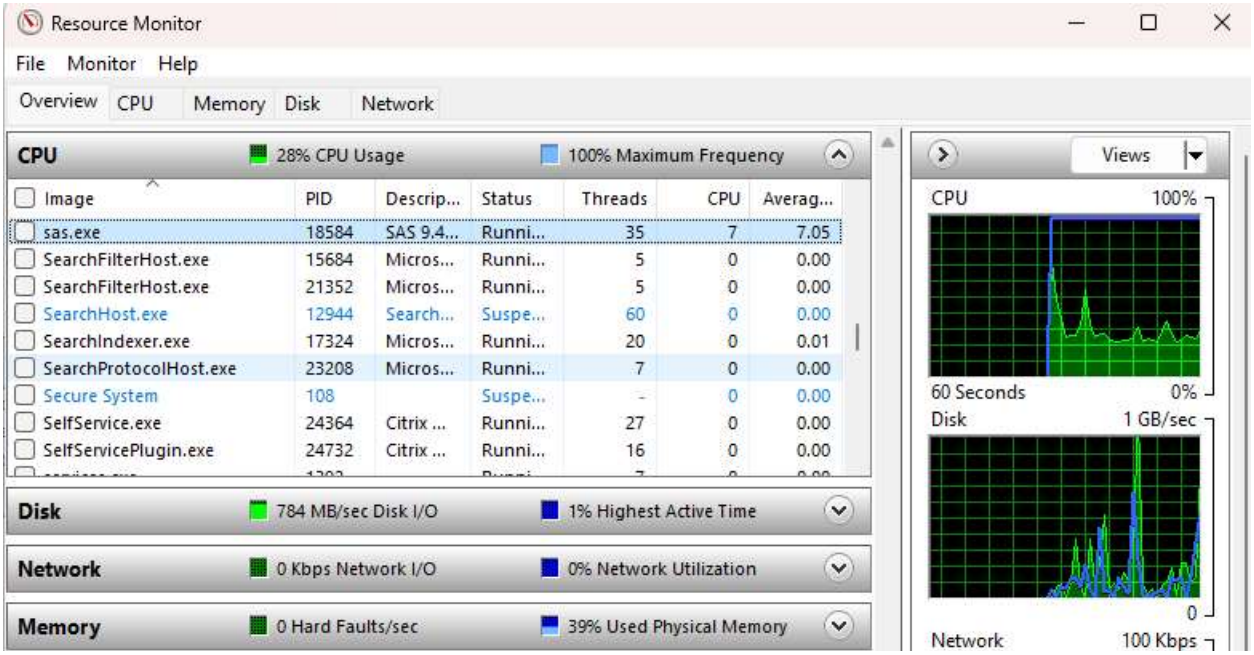
PUTTING IT ALL TOGETHER

Now that you’re armed with knowledge and a few basic examples of techniques readily available to you, as long as you have access to Base SAS, where do you go from here? How do you determine whether the work required to rearchitect your code using parallel techniques is worth it? How do you decide which approach might work? As mentioned previously, if you have access to any of the SAS products designed

for remote execution and parallel processing, dive in there first. Additionally, look for the following “quick wins” that might give you a boost with minor (or even no) code changes:

- MEMSIZE / CPUCOUNT options
- The SPDE libname engine
- Proc FedSQL
- Hardware upgrades on your computer

If you’re prepared to dig a bit deeper, you’ll need to do some research. You have two main ways to evaluate why your code may be taking longer than you think or hope it should: watching a process monitor or studying the code. Using a process monitor is as simple as using either the Windows task manager or the Linux top command and watching the various columns for processor, memory, disk I/O, and network utilization. This should give you a high-level picture of where the bottleneck(s) might be. Digging into the code will help identify the specific pain points and reveal whether and how they might be alleviated. For example, if you see a high network utilization much of the program’s life, look for connections to shared network drives or the downloading of files from remote servers. If you see a lot of CPU resources being used, look for complex data steps or analytic procedures. If memory spikes, look for merges, joins, indexing, hashing, or proc formats. Look for large data steps or file/infile statements if you see heavy disk utilization.



Display 3. Windows resource monitor showing sas.exe information

Determining how effective parallel enhancements might be on your code will take some experience and require you to roll up your sleeves. Practice running your code interactively and applying some basic parallel techniques to one spot at a time, gauging a change’s effect, and tuning the parameters like CPUCOUNT and MEMSIZE as you go. If you have fast disks, breaking up file reads and writes like the above example should make a big difference. If you have a lot of processors and they are underutilized for much of your application, parallelizing larger portions of your SAS code (by putting them into child programs) should make a difference. Network or Internet file transfer is usually not a big win because of bandwidth limitations.

Measuring your improvement

Unless you are in an environment where CPU time costs money (such as with rented hardware or a cloud-based scenario), your only important benchmark should be clock time. To that end, pay attention to the final statement that SAS provides in the log:

```
NOTE: The SAS System used:
      real time           1:27.80
      cpu time            4.04 seconds
```

Determining how long individual steps take interactively and in batch is a bit harder. While procedures and data steps give automatic timing information, macros, SYSTASK statements, or other advanced techniques require inserting a little bit of code. To measure the time it takes between statements, you can use a macro variable to mark a starting point and print the elapsed seconds later. %sysevalf() is required to perform numeric calculations in the text-based macro language.

```
%let _start=%sysfunc(datetime());
** one or more statements **;
%put TIME: %sysevalf(%sysfunc(datetime())-&_start);
```

CONCLUSION

Parallel processing can make a huge difference in the execution time of your SAS programs and, as a result, possibly even your job satisfaction! The world of performance optimization and multiprocessing is fascinating and practical and SAS has provided many products, tools, and language elements to give the SAS programmer many possible avenues to explore. Whether you have access to SAS/Viya with its amazing remote and massively parallel processing capabilities or Base SAS on a PC, you should now be better equipped to think about programming from a more parallel perspective and begin finding optimization opportunities in your own code.

REFERENCES

SAS Institute. 2023. "Threading in Base SAS." Accessed September 19th, 2023 https://documentation.sas.com/doc/en/lrcon/9.4/n0czb9vxe72693n1lom0qmns6zlj.htm

SAS Global Forum. 2009. Langston, Rick. "Scalability of Table Lookup Techniques." Accessed September 19th, 2023 https://support.sas.com/resources/papers/proceedings09/037-2009.pdf

SAS Institute. 2021. Batkhan, Leonid. "Running SAS programs in parallel using SAS/CONNECT." Accessed September 19th, 2023 https://blogs.sas.com/content/sgf/2021/01/13/running-sas-programs-in-parallel-using-sas-connect

SAS Institute. "SAS 9.4 and Container Technology: Build and Run a Container." Accessed September 19th, 2023 https://documentation.sas.com/api/docsets/containers/9.4/content/containers.pdf

SAS Institute. 2023. "What Is Threading Technology in SAS?" Accessed September 19th, 2023 https://documentation.sas.com/doc/en/lrcon/9.4/p0jkap809erpx0n1ww4afz0nl6tg.htm

NESUG 2006. "An Annotated Guide: The New 9.1, Free & Fast SPDE Data Engine." Accessed September 19th, 2023 https://www.lexjansen.com/nesug/nesug06/io/io15.pdf

ACKNOWLEDGMENTS

I want to thank Scott Carl of Tricision, Inc. for his partnership in developing parallel execution tools and his expertise with performance tuning in SAS.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

David Ward
InterNext, LLC
david@internext.io