Structured Tuning: A Layered Approach to Managing System Performance

Steve Wilk & Sam Boles

Hewlett-Packard

Synopsis

You've seen the benefits of Structurism: Structured Design, Structured Programming, Structured Implementation have enabled new achievements in the quality and timeliness of computer systems. Here you'll see the Top-Down principle extended to the perennial subject of System Performance in the form of Structured Tuning.

A team of performance engineers from Hewlett-Packard have developed the Structured Tuning Methodology that starts at the top-level Network and System Layers with the most global factors of system performance -- CPU, main memory and I/O -- and iterates down thru intermediate Layers including Subsystems and Applications dealing with system tables and database control blocks to the low-level Process and Procedure Layers sensitive to individual file placement by logical device and individual algorithms.

You'll see Structured Tuning use a variety of performance tools -- supported, contributed, consultant and proprietary -- to interface with the comprehensive instrumentation integrated with the MPE software, to help you identify performance bottlenecks. You'll see how Structured Tuning protects against Premature Optimization and enables you to focus on high-yield performance efforts. And you'll see how Structured Tuning complements its reactive remedial tools with proactive modeling techniques to enable the Ounce of Prevention Mode.

Background: Techniques, Topology, Tools

The Right Problem

Once in another time and another place, there was a man who had two problems. One of his problems was that he wanted to play a tune on his fiddle. He thought if he solved one of his problems, then he'd have one less problem. So he played a tune on his fiddle, and, sure enough, he got *that* problem solved.

However, in the meantime, *Rome burned.*

He'd solved a problem -- but it was the *wrong problem.*

The Other Cost

We all know there's no such thing as a *free lunch.* Maybe there's no such thing as a *free anything.* To do anything has a *cost.* In fact, it has *two costs.* One cost is what it costs you to do it. The other cost is what it costs you not to do something else. You call that the *opportunity cost.*

A lot of the time it's the opportunity cost that really hurts. You've got a limited budget and limited resources. If you use your limited (and expensive) performance engineering resources to solve the *wrong problem,* the opportunity cost may be that your *real problem* doesn't go away.

The Techniques: A Method to Your Madness

Structured Tuning is a methodology that helps you identify the *right problem* to work on. Structured Tuning decomposes the complexity of a computer system into a series of hierarchical layers that can be traversed from the top down.

Premature Optimization

This top-down discipline works against *Premature Optimization,* one of the costliest parts of the Wrong Problem Syndrome in performance tuning. It forces you to look at the *global issues first,* before getting down to the local issues. If you start at the top and look at the layers in order, you see things in the context of the "Big Picture." This helps you identify the high-yield projects and get your *priorities straight.*

Like Structured Implementation . . .

Structured Tuning does for performance what Structured Implementation does for a project: You identify the main line and build a prototype, fast and cheap. You try it. If it doesn't work because the technology can't stand the strain of the application,

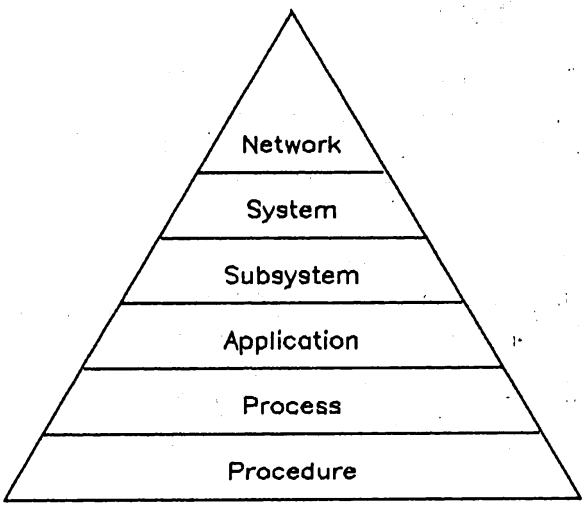
you throw it away. If it's not what the user needs, you throw it away. You use the prototype to solve the *right problem*: *Can it be done? Should it be done?* . . . and if you get a negative on either question, you throw away less code and minimize your losses. If the prototype registers positive, you get into the inevitable rewrite faster and on firmer ground. The Structurism of Structured Implementation helps you ask the *first questions first*; and the same thing happens in Structured Tuning.

Stepwise Traversal

Stepping down thru the hierarchy, if you see at the System Layer that you're bottlenecked on disc I/O, you don't jump right in and rewrite that simulation algorithm (that you *know* is killing you) to cut its CPU consumption in half. That might not be a bad idea . . . eventually . . . but not now. This discipline has Pareto's 80-20 Rule built into it, and gives you a first-order prioritization of your tasks.

The Topology

A dissection of the subject that seems to yield a workable structure for Structured Tuning has the Network as its top Layer and decomposes down thru System and Program Layers to the Procedure Layer:



The Structure of Structured Tuning

Metrics

The "bottom line" is *productivity*. The metrics we use are transaction throughput ("how many orders can I enter in a hour?") and response time ("when I key in 'how much is 2 + 2' how soon do I get the answer after I touch RETURN?")

Tools

To get these metrics in this range of layers there's a range of tools. Some are software products you can buy. Some are contributed. Some are part of your HP Performance Specialist's bag of tricks. These tools for the most part interact directly or indirectly with the comprehensive performance measurement instrumentation that is part of MPE.

A key measurement tool in the HP3000 domain is OPT (On-line Performance Tool). This tool is a supported product and provides a variable-interval, variable-level window into the system.

Complementing OPT is a consulting tool MPEDCP (MPE Data Collection Program) which collects performance statistics similar to those in OPT, but with a somewhat different focus and format.

For finer granularity at the file/process level, IODCP is a consulting tool that gives statistics to help with mid-range and low-level tuning.

At the Process and Procedure Layer the APS (Application Program Sampler) system provides insight into CPU utilization down to the level of lines of code.

In addition to these tools, there are MPE and subsystem commands and utilities that enable views of how the system's running, eg SHOWCACHE, DSCONTROL, DSDUMP.

Modeling

Complementing the performance monitoring tools are the models of the "real world" that can give valuable support to Capacity and Configuration Planning and Performance Management: Benchmarking, Simulation, Analytic Modeling.

The Ounce of Prevention Mode

While the monitoring tools tend to be reactive to problems that already exist and need to be corrected, models are more predictive and proactive. With predictive power, models can provide the information necessary to make a good decision at the beginning that can prevent a problem rather than fixing it after it's happened. This *Ounce of Prevention Mode* can save money, time and human resources.

Benchmarking

In Benchmarking, the first degree of abstraction from the "real world," we model the range of system and workload configurations that is proposed or perceived to be a correct representation of the current situation and/or the future. Being a model only a step away from the real thing makes Benchmarking tend to be very costly in time and other scarce resources.

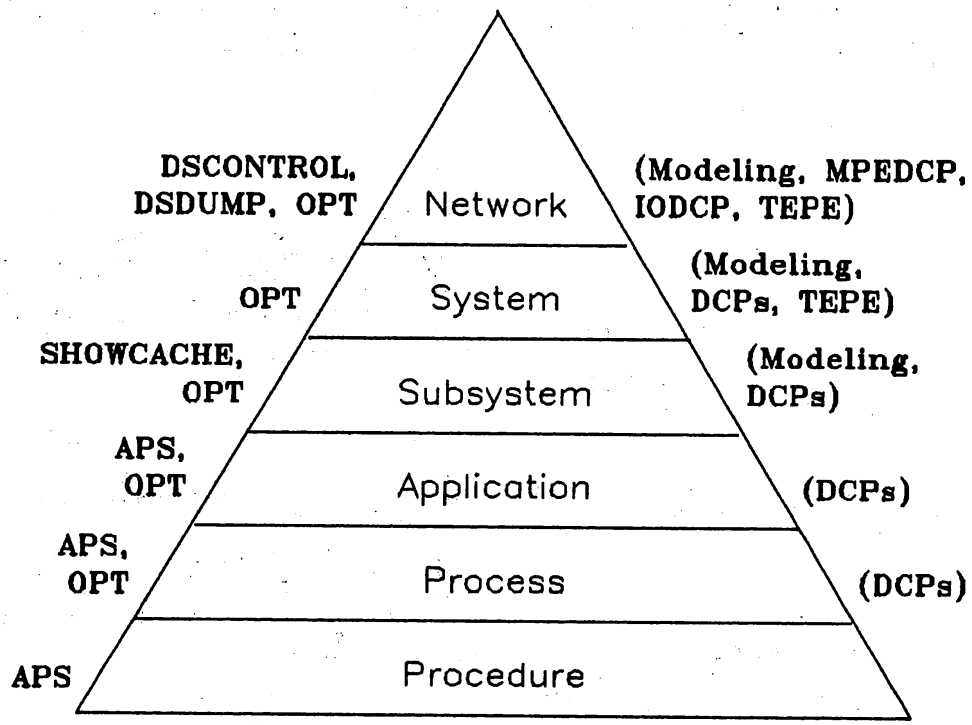
For benchmarking, TEPE (Terminal Emulator and Performance Evaluator) provides a direct hardwire connect from the ADCC on the driver system (Series 4x) to the ADCC or ATP on the System Under Test (Series 4x or 6x). Guided by terminal "scripts" TEPE sends bit-serial messages that look to the HP3000 under test like inputs from CRT's direct-connected to it.

Simulation

The second degree of abstraction from the "real world," Simulation, models a trace of the "real world"; for example, a detailed sample of disc I/O traffic, taken from a production system is modeled to predict its behavior in a proposed environment, eg a system where disc domains are cached in main memory.

Analytic Model

The third degree of abstraction from the "real world," the Analytic Model, models algorithmically the functions of the "real world"; for example, a queueing-theory based central server network that includes a CPU of variable speed and a variable number of discs with variable speeds and variable user classes and variable think times and variable . . . and variable . . . This model is typically the fastest and cheapest, can be the most flexible, is sometimes the only feasible, but often is the most difficult in some ways since it requires us to *understand our data and programs and systems in great depth.*



The Mapping of Tools in Structured Tuning

System Components

CPU

The Central Processing Unit is a key variable in the Structured Tuning Methodology, but in the typical commercial job mix it's usually not where you get the primary performance bottlenecks. In the current HP3000 line you have a .4 MIPS processor in the 4X series and a 1.0 MIPS processor in the 6X series for vertical flexibility. For horizontal flexibility, the HP DSN (Distributed Systems Network) provides a variety of 3000-to-3000, 1000-to-3000, and 3000-to-other linkage possibilities.

CPU and Disc Caching

Any performance tuning methodology today in the mid-range to high-end of the HP3000 must examine the implications of Disc Caching. This may be the most significant performance contribution in the HP3000 family since the 1 MIPS processor; it makes something beyond a brute-force raw-horsepower performance improvement. Disc Caching needs 20-40% of the CPU in order to give you really substantial (2X, 3X) performance gains.

If the necessary CPU is not available, Disc Caching can degrade performance.

Main Memory

In the HP3000 family you've got this range of main memory available:

Series	Min	Max	Write Speed
68	2Mb	8Mb	375 ns
48	1Mb	4Mb	530 ns
42	.5Mb	3Mb	530 ns
39	.5Mb	3Mb	530 ns

On the high-end, in the 6X series, you have an 8K byte high-speed (ECL RAM) memory cache that has a 95% hit rate. These gives you an effective 145 nsec memory read speed.

Main Memory and Disc Caching

Let's put main memory in the Disc Cache perspective: The essence of Disc Caching is to let you get at your disc data at main memory speeds rather than disc access speeds. If you compare the two, you find that accessing main memory is something like 10000X to 50000X faster than your average disc memory. Since disc I/O makes up such a large part of commercial data processing, that kind of data access improvement can really pay off in better response time and throughput rates. If your job mix has a 70% cache hit rate (as is often the case in commercial applications, you can pay the 3 or 4 msec price for Caching, save 30 or so msec on 70% of your I/O, and get a good return on your investment. To give Disc Caching room to work, you need to give it 1 or 2 Mbytes of main memory on the high-end machines to get the 2X and 3X performance gains it's capable of producing.

If Disc Caching is enabled on a system already under memory pressure, it may degrade performance.

Disc Memory

Disc I/O has been the perennial performance bottleneck in the typical HP3000 job mix -- or *so it seems.* With the arrival of the 4X and 6X machines, the old 30 I/O's per second max has gone away. With multiple master drives (each with its own controller), on multiple GIC's (General I/O Controller) (each with a 1 Mbyte bandwidth), on multiple IMB's (InterModule Bus) (each with a 3 Mbyte bandwidth), we can sustain 60 to 120 or more disc I/O's per second.

But we don't.

Why?

Because *we don't ask for it.*

What we usually do is port our jobs 1-for-1 from the Series III to a 44 or a 64, configure in 2 or 3 times as many terminals as we had before to go against that single IMAGE database that we built 4 years ago when we were 1/4 the size we are today. And we sit there, queued up on a single-threaded Database Control Block with the CPU only 50% Busy and a disc subsystem capable of 100+ I/O's a second being asked to do 50 I/O's a second.

Disc Memory, Disc Caching and PEP

For the high-end 3000 installation, one of the important disc drive innovations in recent years is the 400 Mbyte HP7935.

When we first heard about the 7935 we were really impressed: for a small percent more than you pay for a 7925 you could get more than 300% of the 7925 capacity. Then we ran some benchmarks and found that for the capacity gain you paid a 15-20% performance penalty.

Then came PEP (Performance Enhancement Project.) The engineers in Boise tweaked better than 15% improvement into the 7935 and got it within noise-level of the 7925 performance.

The PEP microcode structure gain coupled with Disc Caching, Buffer Prefill and RPS (Rotational Position Sensing) makes the on-line storage economy of our 400 Mbyte disc drives (7933, 7935) a viable solution

for the performance-sensitive customer whose "results must be measured by performance."

But remember: Disc Caching needs file locality, a good (3+:1) Read:Write ratio and a heavy disc I/O load to produce the 2X and 3X performance gain it's capable of.

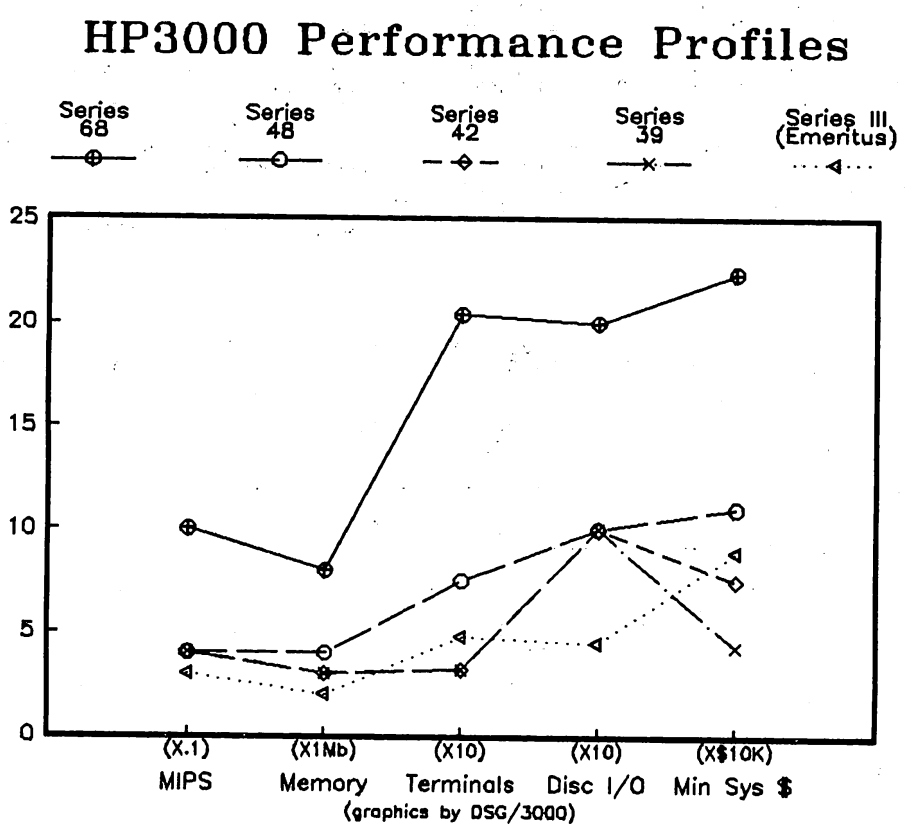
Other System Components

In addition to the Big Three -- CPU, main memory, disc I/O -- your system performance can be influenced by factors like:

- Datacomm line speeds for terminals and distributed systems
- Terminal/Workstation speed and intelligence
- Mag tape speed/density (especially in back-ups)
- Printer capacity
- System tables

Structured Tuning looks at these components as they emerge at the various Layers of the hierarchy and assigns them a direction and magnitude appropriate to their station in life. This is usually very application- or installation-dependent.

Here's an "artist's conception" of the performance characteristics of the HP3000 family:



Structured Tuning in Action

Now you've seen the topology of Structured Tuning, the objects that it deals with and some of the tools that it uses. Let's take a look at some examples of Structured Tuning in action.

The Network Layer

In some ways, this one may be the toughest. You've got a wide range of configurations, line speeds, ranges, types, sharing, simultaneity, concurrency, costs, metrics, vendor compatibility issues.

From the aspect of Structured Tuning, there are some rules of thumb that seem to work. Aim at doing fewer things bigger. From the performance aspect, one user with a large record is better than multiple users with several small records. One thing you can check for is a small overrun on your record size; for example, if you configure 1024, but send 1026 you may generate a lot of continuation records.

If at the System Layer you find a CPU bottleneck that traces to the Application Layer where you find that the heavy crunch of business graphics is consuming your CPU at the expense of general response time, consider relieving the CPU pressure by distributing the load. For example, an RS232 link to an HP9000 could enable the offloading of the heavy-duty processing to a high-performance, high resolution work-station solution. The artwork database in a convenient intermediate form could then be uploaded to the HP3000 for integration with text and output to the laser printer.

Notice the Structured Tuning Methodology at work here: it iterates thru lower Layers for details characterizing problems detected at a higher Layer, and may percolate up to the highest Layer for the solution.

The System Layer

A good starting point at this Layer is to look into the system thru the window that is OPT (On-line Performance Tool). At the global level, OPT shows you CPU, main memory and Disc I/O utilization on a variable interval basis, and CPU and Disc I/O utilization on a cumulative (multi-interval)basis.

Here's an example of the OPT global report (clipped on the right for printing convenience):

```
      RESOURCE USAGE DISPLAY      HP32238X.VV.16  OPT/3000
(C) HEWLETT-PACKARD COMPANY 1979, 1980                        SEBDC0L2 S64 2MB CACHE ON
   CURRENT INTERVAL:     66.5 SECONDS          OVERALL INTERVAL:     11.1 MINUTES
ACTIVITY IN CURRENT INTERVAL
                          10        20        30        40        50        60
      MEMORY USAGE M----MC----------CS----SD--DK-----------------------------------

         CPU STATE B-------------------------------------------------------BP-------

                          10        20        30        40        50        60
 DISC I/O ACTIVITY SSK-----------------------------------K

-------------------------------------------------------------------------------
ACTIVITY OVER ALL INTERVALS
                          10        20        30        40        50        60
         CPU STATE B-------------------------------------------------------BP-------

                          10        20        30        40        50        60
 DISC I/O ACTIVITY SK------------------------------------K

MEMORY USAGE LEGEND:
  M  Resident MPE
  C  Code segments
  S  Stack segments
  D  Data segments
  K  Cached disc domains

CPU STATE LEGEND:
  B  Busy on processes
  P  Paused for user and/or memory management disc I/O
  I  Idle
```

```
  G  Garbage collection
  O  Memory allocation and ICS overhead

DISC I/O ACTIVITY LEGEND:
  U  User disc I/O
  S  Segment management  I/O
  K  Cached I/O
```

Here we may see excess CPU, main memory, disc I/O capacity, Pause for I/O.

If we have a disc I/O intensive load, we can drop to the next Layer in our Structure, the Subsystem Layer, and take a look at the disc subsystem.

The Subsystem Layer

Here we can collect a trace of the disc I/O traffic and simulate what its behavior would be with Disc Caching.

We can look at the read:write ratios. We can see if the traffic is sequential or direct. We can look at file access locality in time and space. And we can run the trace thru the Disc Caching Simulator to see what kind of hit ratios we'd get if we enabled Caching and gave it a Mbyte or so of memory to work with.

If it looks promising, we can put in a Meg of Main, enable Disc Caching and take a look at what it does for performance:

```
:SHOWCACHE

DISC    CACHE     READ WRITE          PROCESS              % OF    CACHE
LDEV  REQUESTS    HIT%  HIT%  READ%   STOPS     K-BYTES  MEMORY  DOMAINS
-------------------------------------------------------------------------

1       16436       70    93     73    3776        1434      23      252
2       17809       69    95     66    3673        2319      38      358

Total   34245       70    94     69    7449        3753      62      610

Data overhead is 158K bytes.
Sequential fetch quantum is 96 sectors.
Random fetch quantum is 32 sectors.
```

We see we get fewer process stops (the Cache Manager does not interrupt on a cache hit for a read), get bigger and more level physical I/O's, put to use the idle CPU and main memory and, by happy coincidence, double the throughput, cutting the wall time about in half.

The Application Layer

For the next case study of Structured Tuning in action, let's look at what made it feasible to print multiple copies of the paper you're reading on an HP2680 Laser Printer. Let's say we have a technical publication that we've typeset on the laser printer, using 3 forms and 12 fonts and we've decided to print 120 copies of it on the system. We did a TDP (Text and Document Processor) "Final to *2680" to get TDP to format it, deferring the SPOOL file. Then we did a "HEADOFF 14" (to save a tree), set OUTFENCE up so we could get the right fold on the document and not get alien output commingled with our 120 copies. We then did an "ALTSPOOLFILE #Onnn;COPIES=120" and set the priority over the fence.

It's a 10-pager, so we go away and decide we'll come back in (10 x 120)/45) minutes. (We've got 120 copies of a 10-page document running on a 45 page-per-minute laser printer). We come back in about a half hour, smile pleasantly as we walk through the scowling crowd waiting impatiently around their/our laser printer. We reach for our 120 copies and find we have only 30.

We pull out our Structured Tuning notes, at the System Layer we run OPT, get the global display, find heavy disc I/O. We go into the Process display, see that the SPOOLER is doing a lot of I/O. We run SPOOK, find 1100 lines of dense environment file at the beginning of our output.

We then observe the laser printer. At the end of each document, as our heavy-duty environment file is downloaded, because of the size of the file, the laser printer infra-red fuser does a time-out, then goes into a warm-up cycle. This cuts our printer performance to 10 pages a minute.

We smile sheepishly at the irate by-standers, as we set our SPOOL file priority to 1, turn the Headers back on and reset the OUTFENCE to normal.

We then create a STREAM file to:

run SPOOK
text in our SPOOL file
APPEND the environment file once
APPEND 10 copies of the text
to the environment file
APPEND END to get a new SPOOL file

We run this in background, so no one gets impacted in response time.

When we get the new 10X SPOOL file, we go thru the HEADOFF etc routine we did before, set copies=12, and the printer runs at about 38 pages a minute rather than 10.

Structured Tuning at the Application Layer helped us isolate the problem. Structured Implementation built this *ad hoc* prototype to see if the technology could do the job. (The next step in the Structured Implementation Methodology is expected to make the job operationally feasible and give a 2X to 3X overall performance improvement.)

The Process Layer

Let's say that in our Top-Down Traversal of the Structured Tuning Structure, we establish at the System Layer that we have a performance problem that we trace thru the Subsystem Layer to the Application Layer. And let's say that the application has multiple users running the same program, and we think it's one of the processes of that program that's causing our problem.

We can get into OPT and go to the process display to try to further isolate the problem.

The Procedure Layer

If it turns out that our top-priority performance problem is traceable down thru the hierarchy to a given process, we may want to look at CPU consumption at the procedure level or even at the granularity of lines of code. APS (Application Program Sampler) can help us here and enable us to pinpoint the sector where the heavy resource consumption is happening. With this focus and granularity, we can have a good calibration on where we can best utilize our limited performance tuning resources.

In the APS CPU Utilization report below, we see that the GEN'CHAR procedure (which does a 2-dimensional raster scan of a character "ROM" within a 2-dimensional scan of a message list) is probably a good place to tune among the 10 procedures in the program -- 8 of which didn't have enough activity to register on the scale.

Conclusion

You've seen how the Structured Tuning Methodology is an iterative process that starts at the topmost global layer. It can help you keep a balanced perspective as you look at performance problems. You've seen some of the tools and how they're applied at the various layers to zero in on bottlenecks. And you've seen how Structured Tuning can help you find and fix the *right problem.*

Program Name: SEBPROG.BOLES.CAD - Segment: %301 SEG'
Distribution of Direct CPU Time Over Procedures
(95 Segment Samples = 5.26% Direct of Program Time)

```
+-------------|---------|---------|---------|---------|---------|+---%---%CUM--
SET'UP'CHAR'ROMDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD | 45.3  45.3|
|   GEN'CHAR DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD| 54.7  100.|
+-------------|---------|---------|---------|---------|---------|+---%---%CUM--
Minimum bar threshold is .0%
```

APS CPU Utilization Report

About the Authors ....

Steve Wilk *is the Performance Center Manager at the Hewlett-Packard computer facility in Cupertino, California, with responsibility for performance benchmarks and product characterization. His 18 years in the industry cover a wide range, including programming, systems analysis and program management. With an undergraduate degree in mathematics, Steve received his MBA from the University of San Francisco.*

Sam Boles *is a Systems Engineer in the Systems Performance Center at the Hewlett-Packard computer facility in Cupertino, California. With HP for seven years, Sam's computer experience started back in the AUTOCODER days of the 1401/1410, migrated thru the 360/370 era, and now focuses on HP hardware and software performance characterization. Sam earned his MS at UCLA in Information Systems.*

sebiug37 2345 14Nov83