SYSTEM FUNCTIONAL MIGRATION - HIGH LEVEL LANGUAGE TO HARDWARE

Y. S. Wu and G. R. Lloyd
Naval Research Laboratory
Washington, D. C. 20375

Abstract

This paper intends to examine system functional allocation among available processing resources in a systematic way. The discussion will focus first on the criteria for assigning functions to processors, that is, horizontal migration. Then the implementation of "trade-off" choices of implementing a given algorithm in a particular processor in either high level language, assembler language, microprogram, or hardware will be examined (vertical migration). A quantitative measure, based on duty cycle, file size, and program size of the functional algorithm, is derived for vertical migration.

Introduction

Computer science is not a science. A decade to five years ago this author may have been a voice crying in the wilderness. Of late, some big name computer science gurus have begun to make the same assertion. It is at best a pseudo science, embracing metaphysical principles and perhaps scientific methodology to deal with human behavior and resource management in the materialistic world of gadgetry. It is fitting and proper to rename the field 'computer sociology'. In the soft sciences it is fair to invent metaphysical laws (named after the inventor) when a scientific theory cannot be rigorously derived or proven. Some of these well known laws include: Pareto's Law, Parkinson's Law, Murphy's Law, and, of late, Brooks' Law (not to be confused with Brooks' Bill). Although this author is not a computer scientist, he is willing to follow in the footsteps of these celebrities and invent his own metaphysical principle, Wu's Principle, which paraphrases and contradicts the famous Peter Principle. Wu's Principle states:

> In computer science and software, everyone grossly exceeds his or her level of incompetence.

The point of this paper is that the elegant solution to self destructive software and regeneratively complex problems is most often provided by an intuitive, simplistic and common sense approach, rather than uncertain analysis.

Pareto's Advice to Programmers

Many savants recommend the use of high level programming languages to control the cost of producing and maintaining software. The most common objection raised to the use of high level languages is the cry of 100% efficiency. This is very foolish. The nineteenth-century Italian economist Pareto observed that the relationship between utility and volume is 80/20: 80% of the value is derived from 20% of the transactions. In management, 80% of the useful information is derived from 20% of the total paper flow. The advent of the Xerox machine and the electronic computer have simply altered the percentages: the modern ratio is 90/10. In programming terms at least 90% of the software for a system should be coded in a high level language; the remaining 10% might be machine dependent or so heavily used that a faster and more expensive implementation technique might be used.

This observation should have an important influence on the system designer. Solve the problem using all the tools of high level languages before building esoteric hardware or cluttering up a microprogrammed processor with useless (but interesting) special purpose opcodes. At the very least, the designer might learn what the system <u>really</u> must do and where bottlenecks occur.

The first round of mistakes should be as inexpensive and educational as possible. If the rules stated below are followed, different implementations of the same functions can be substituted as required without disturbing parts of the system which appear to work properly. If disaster strikes, it is a software disaster (par for the course) rather than a system disaster which can be very damaging to the professional pride of the designer.

Migration

When a software system designed to run on a single processor exceeds the computing capacity of the processor, two simple courses of action are open. The first is to offload the overworked processor by introducing one or more functionally dedicated processors to handle part of the computing load. The second is to speed up the execution of the critical functions by introducing carefully written assembly language routines, microcode, or additional hardware support. In both cases, the high level language viewpoint can be profitably applied to simplify the programming of the entire system.

If high level languages are good for software cost control they should also be used as a basis for simple system design. When forced to use multiple processors and multiple levels of system implementation techniques,

only two high level language features are used to build the system control structure: procedure call and process to process message passing. The programmer should not care how or where a procedure call is executed or the location of the process receiving or transmitting a message. Let the runtime environment do the schlepp work.

Horizontal Migration

If the critical functions are well defined and have specialized requirements (as in signal processing) the obvious approach is to introduce a dedicated processor which performs a few functions very efficiently. All of the work to be done is defined by the procedure name and the parameter values passed by the call or by the content of the process to process message. As a rule of thumb, communications overhead should be less than 10% of the processing time gain achieved by farming out the process to a subprocessor (G&A for managing a subcontract should not exceed 10% or someone deserves to be fired).

Vertical Migration

Vertical migration chooses an implementation technique within a single processor. Typical levels of implementation are high level language, assembly language, microcode in writable control store (flabbyware), microcode in fixed control store (firmware), and hardware. The rule for determining how different functions are to be implemented is really a rule for calculating return on investment. Simple algorithms which execute in tight loops are the best candidates because minimal investment in specialized implementation can have a high payoff.

W-Test

The following rule has been successfully applied in the design of complicated signal processing systems, where functional routines are well defined arithmetic kernals. It is also applicable where the functions are non-arithmetic but well defined (such as operating system kernals).

Let:

D = functional routine duty cycle (fraction of total load)

A = number of elements in operand arrays (number of data points)

P = number of instructions in functional routine

Then the migration figure of merit W is:

$$W = A*D/P$$

Normalize W=1 when total data and program memories are consumed with 100% execution duty cycle within the processor.

By applying Wu's conjecture of 90/10, the following functional migration rules of thumb are derived:

i) program in high level language when W is less than 0.1
ii) program in assembly language when W is 0.1 to 1 and D less than 0.1
iii) microprogram when W is greater than 1 and less than 10, or when W is 0.1 to 1 and D greater than or equal to 0.1
iv) implement in hardware when W is greater than 10

The W-Test is dangerously simple. Given a hand calculator, even a bureaucrat can find some embarrassing tradeoffs in system design.

Conclusion

This paper is written in simple terms for all levels of management. The key observation is that the optimal solution to an undefined problem is not a complex kludge, but rather no solution at all. A manager may encounter cases where no data is available to apply any of the recommended tests and tradeoffs. Here the course of action is very straightforward: hire a new staff.

For a migration example, see:

A. van Dam, G. Stabler, and R. Harrington, "Intelligent satellites for interactive graphics," Proc. IEEE, vol. 62, no. 4, pp. 483-492, April 1974.

The following pages of visual aid texts were given in support of this paper (Ed.)

SYSTEM FUNCTIONAL MIGRATION

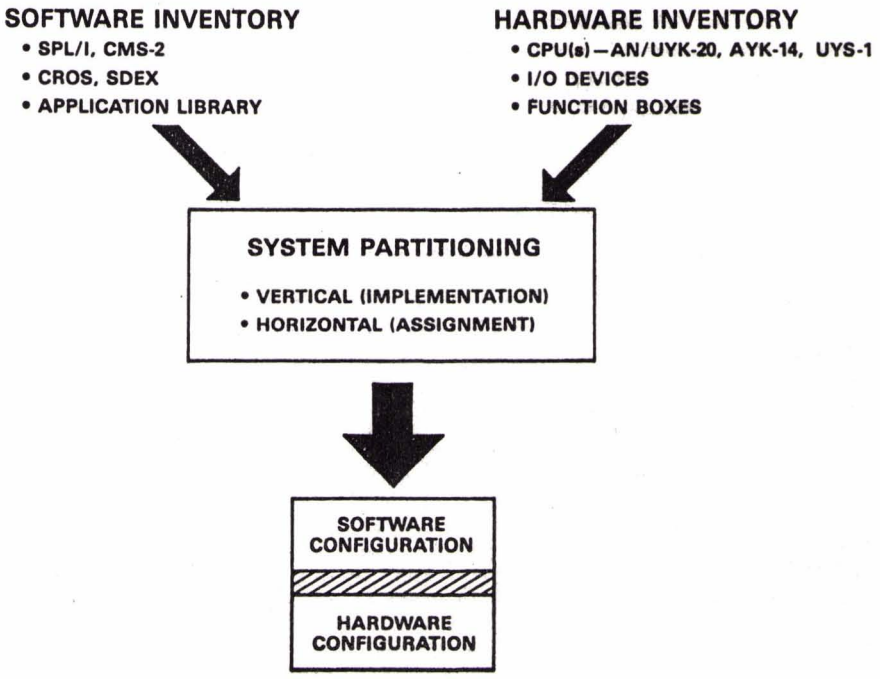
HIGH LEVEL LANGUAGE

TO

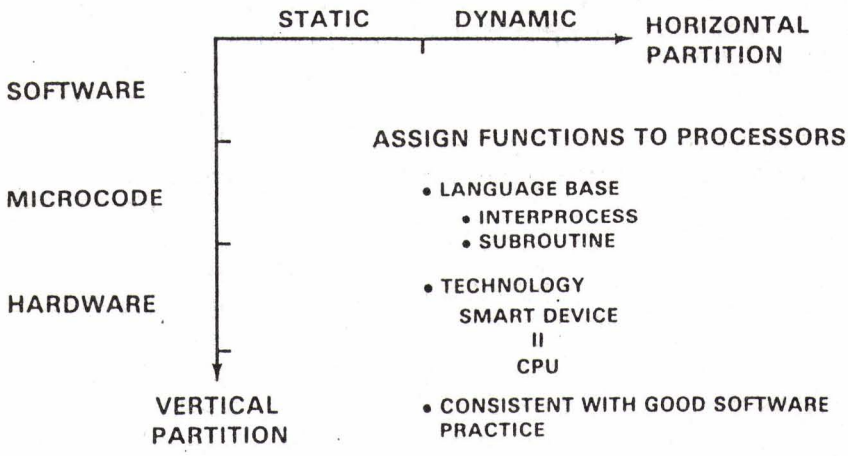
HARDWARE

WU's PRINCIPLE

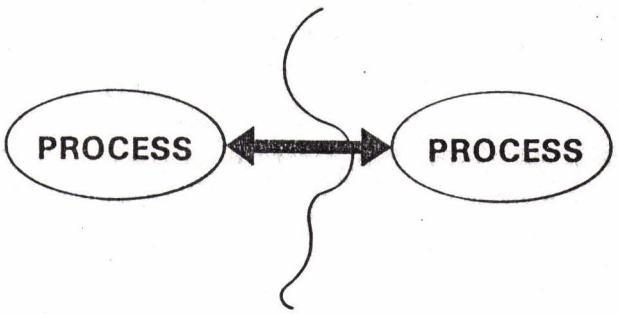
IN COMPUTER SCIENCE AND SOFTWARE, EVERYONE GROSSLY EXCEEDS HIS OR HER LEVEL OF INCOMPETENCE.



HORIZONTAL MIGRATION

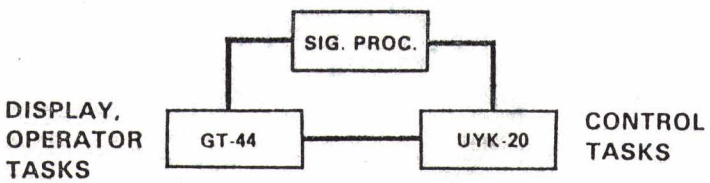


STATIC ASSIGNMENTS

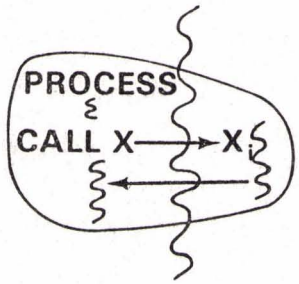


HORIZONTAL PARTITIONING EXAMPLES

PROCESS ⟷ PROCESS (STATIC) SIG. PROCESSING TASKS



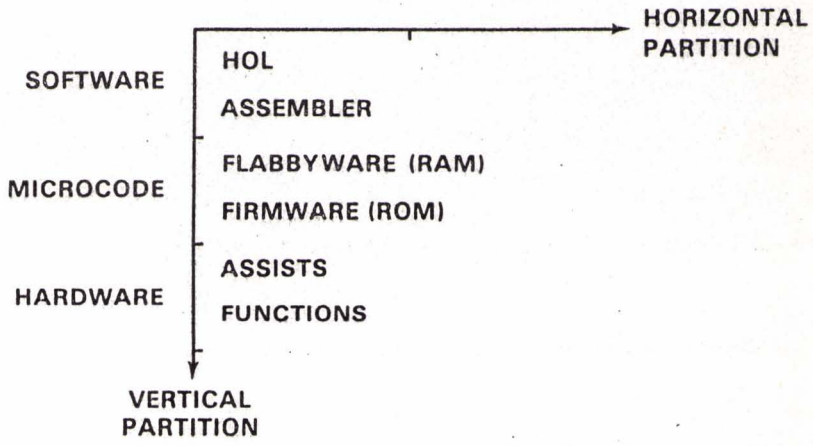
DYNAMIC ASSIGNMENT



EXAMPLE:

SPL/I CONTROL OF DISTRIBUTED
SIGNAL PROCESSING ARCHITECTURE

VERTICAL MIGRATION



CHOOSE AN IMPLEMENTATION APPROACH:

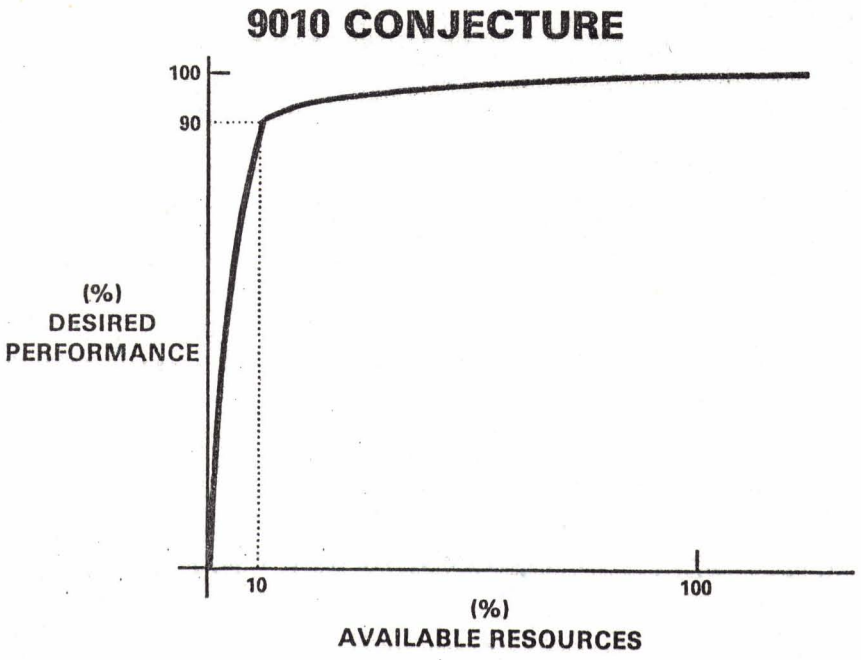
- THROUGHPUT REQUIREMENTS
- DEVELOPMENT COST
- MAINTENANCE COST

JUSTIFICATIONS FOR MICROPROGRAMMING

- HIGH DUTY CYCLE
- SMALL KERNEL
- LONG DATA ARRAYS

WU'S CONJECTURE OF 9010

- EXTENSION TO PARETO'S LAW
- SOCIOLOGY
- ECONOMY
- BUREAUCRACY
- HARDWARE
- SOFTWARE



9010 CONJECTURE FOR SOFTWARE

- 90% OF THE PROGRAMS RUN 10% OF THE TIME
- 10% OF THE PROGRAMS RUN 90% OF THE TIME

ω-TEST

LET:

DC = FUNCTIONAL ROUTINE DUTY CYCLE

|A| = NO. ELEMENTS IN OPERAND ARRAYS

|P| = NO. INSTRUCTIONS IN FUNCTIONAL ROUTINE

THEN:

$$\omega = \frac{|A|}{|P|} \times DC$$

ω NORMALIZATION

$\omega = 1$

WHEN TOTAL DATA AND PROGRAM MEMORIES ARE USED WITH 100% DUTY CYCLE

MICROPROGRAM

WHEN:

(A) $\omega > 1$

(B) $1 \geq \omega \geq 0.1$ AND $DC \geq 0.1$

ASSEMBLER PROGRAM

WHEN $1 \geq \omega \geq 0.1$ AND $DC < 0.1$

HLL PROGRAM

WHEN $0.1 > \omega$

HARDWARE IMPLEMENTATION

WHEN $\omega > 10$

ASP EXAMPLE

FFT:

$|A| = 1{,}024$

$|P| = 70$

$DC = 0.2\ (20\%)$

$$\omega = \frac{1{,}024}{70} \times 0.2 = 2.9$$

RECURSIVE FILTER:

$|A| = 512$

$|P| = 60$

$DC = 0.4\ (40\%)$

$$\omega = \frac{512}{60} \times 0.4 = 3.4$$

MORE ASP EXAMPLES (Continued)

HARDWARE IMPLEMENTATIONS

"BUTTERFLY"

VECTOR AND FFT ADDRESSING

DeMUX

HORIZONTAL MIGRATION RULE OF THUMB

COMMUNICATIONS OVERHEAD SHOULD BE LESS THAN 10% OF THE PROCESSING TIME GAIN ACHIEVED BY FARMING OUT THE PROCESS TO A SUBPROCESSOR

"It must be remembered that there is nothing more difficult to plan, more doubtful of success, nor more dangerous to manage, than the creation of a new system. For the initiator has the enmity of all who would profit by the preservation of the old institutions and merely lukewarm defenders in those who would gain by the new ones."

Machievelli, "The Prince", (1513)