DEVELOPING SPPVM MODULES WITH VISUAL BASIC

MOSES E. EKPNYONG, NNAMSO M. UMCH AND ETIM E. EKONG
(Received 16 August 2001; Revision accepted 5 September 2002)

ABSTRACT

The development of true Parallel Machines of which literatures are scanty is still algorithmic. Since the need for increasing power in computations is the wish of every programmer or designer, PVMs become more of a choice. PVMs enable large computational problems to be solved more cost effectively by using aggregated power and memory of many computers. This paper develops a SPPVM (Single Processor Parallel Virtual Machine) with Visual Basic. It employs the VB Shell command to cause a single program or instruction to execute in two different shells in same memory space. A quick sort program is implemented on two shells. Each shell sorts 500 elements. The machine used for the execution is: Intel (r) Celeron™ Processor – Genuine Intel ~ 600 Mhz, with Total Physical Mem. = 30.45MB, Available Physical Mem. = 232KB, Total Virtual Mem. = 2GB, Available Virtual Mem. = 1.90GB, Page File space = 1.97GB at runtime. Result comparison with a strictly sequential version reveals that the SPPVM executes in less time than \( \frac{1}{2} \times T_{seq} \) (the theoretical value for two processes in parallel). The key benefit of this paper is to enable programmers explore parallel programming features on micro systems and develop their own SPPVM.

Keywords: Shell function, Self-diagnostic, Multi processing, Multi-Platform, Parallel Processing.

INTRODUCTION

PVM (Parallel Virtual Machine) is a software package that permits a heterogeneous collection of Unix and/ or NT Computers hooked together by a network to be used as a single large parallel computer. The software is very portable. The source, which is available through NETLIB, has been compiled on everything from Laptops to CRAYS.

Current research on PVM reveals that PVM 3.4.3 Release includes self-diagnostic install and input from RedHat Linux and NASA for improved use on Beowulf clusters. New features in PVM 3.4x include communication contexts, message handlers, persistent messages, and interoperability between NT and Unix clusters, (See Kung, 2001).

SPPVM DESIGN

SPPVM design is similar to multiprocessors. Multiprocessing allows more than one program to be in states of execution at any given time. Each program executes until it is blocked, at which time control of the CPU is passed to the next process (Davis, 1996).

In Microsoft Windows ‘9x, etc. each process has its own memory space to prevent one
process from interfering with the others. SPPVM is similar except that the different processes execute in the same memory space.

The various versions of Windows can Multitask, or run several applications at once. (How effective this will be depends upon how the applications were written. Modern Windows Applications can be multitasked by the OS alone; applications designed for Windows 98 must cooperate by relinquishing control for multitasking to work.) Visual Basic offers this advantage of Windows Multitasking capabilities by accepting code that activates any Windows Applications or that sends commands directly to the active application from a Visual Basic Project.

Graphics and Text can be exchanged between Windows Applications in Visual Basic using the Clipboard. The Clipboard can be used together with the properties given in Cornell, 1998 to implement similar features in projects. The Clipboard can only hold a piece of related data at a time. A sample program applying the Clipboard command is shown in Cornell, 1998; pp 750 – 752.

Chapell, 1996 overviews COM/OLE. Visual Basic applies the Active X/COM/OLE properties and allows programmers build integrated Windows Applications using Visual Basic as a “glue” to bind disparate objects and applications together. (The objects can be accessed not only on a single machine but also on the Internet).

VB Shell function runs any .com, .exe, .bat or .pif files from a Visual Basic Program. For example, a PASCAL Program can be called under Windows 98 with a line of this nature:

Shell “C:\PASCAL\PASCAL.EXE MYPROG.PAS”.

The OS must know where MYPROG.PAS is located. This is possible if the file being “Shelled” to is located in a directory in the path or in the current directory. When a program is shelled, a new iconized Window is generated and given a focus. From this principle; heavy computational jobs can be split into stratified programs or modules, shelled and allowed to execute virtually in parallel. With this, Windows, Visual Basic, Currently running program, and a “shelled” program can simultaneously run in memory, otherwise, we may have to rely on Windows to manage the memory by swapping to disk, but execution will slow down dramatically. However, it is far more cost effective to expand the memory than to build a parallel machine.

SPPVM MODULES

The concept of parallelism centres on several common parallel programming models, which include: pipelining, data partitioning, recursive computations, domain decomposition, divide-and-conquer and multi-functional pipelining. Brief descriptions of these can be found in Kung, 2001. A complete example program written in W2 is presented below:

module Stepdata (a in, b out)
float a[1000], b[1000]
cellprogram (cellid:0:9)
begin
    function step
    begin
        int j;
        float temp;
    end
    for i = 1 to 1000 do
        temp = a[i] + b[i];
        b[i] = temp;
    end
for j := 0 to 999 do 
  begin 
    receive (left, r, temp, a [j]);
    send (right, r, temp + 1, b[j]);
  end; /* of for statement */
end /* of function step */
call step;
end /* of function step */

In the above program, each cell receives data from its left neighbour, adds one to each data item before passing it to its right neighbour. W2 is a single Pascal-like high-level programming language for the Warp array. W2 hides the low-level details of the Warp Computer and provides a high-level abstraction for the Warp Programmer. Warp modules can be called from a C Program running on the host. This is done through a well-defined set of functions in Warp User Package.

Assuming three Pascal programs are saved with the following names: PSAMP1.PAS, PSAMP2.PAS and PSAMP3.PAS in a directory PASCAL of which PASCAL.EXE is found. Then a SPPVM Module that executes these programs virtually in parallel with VB is as shown below:

Private sub SPPVM_module()
  <System settings>
  <Interface design>

  Shell "C:\PASCAL\PASCAL.EXE \PASCAL\PSAMP1. PAS" 'Shell Program 1
  Shell "C:\PASCAL\PASCAL.EXE \PASCAL\PSAMP2. PAS" 'Shell Program 2
  Shell "C:\PASCAL\PASCAL.EXE \PASCAL\PSAMP3. PAS" 'Shell Program 3

  < Other Command >

End sub

With this model, n programs could be shelled and run in same memory, provided the compiler and the executing program paths are correctly specified.

It is also possible to shell different programs running on different compilers.

We here present a complete VB Program (a SPPVM Module) with two shells; each shell executes the same program and outputs the resultant time. A sequential version is also presented. The data used for the illustration are generated randomly. This program serves as an execution platform. With this idea, the reader can design stratified programs in any language of his choice, compile them to executable files and modify this module to suit his calls.

VERSION 5.00
Begin VB.Form sppvm_module
Caption = "Program Window"
ClientHeight = 3195
ClientLeft = 60
ClientTop = 345
ClientWidth = 4680
LinkTopic = "Form1"
ScaleHeight = 3195
ScaleWidth = 4680
StartPosition = 3 'Windows Default

Begin VB.CommandButton Exit_command
    Caption = "Exit"
    Height = 375
    Left = 1680
    TabIndex = 4
    Top = 4560
    Width = 1095
End

Begin VB.Frame Frame1
    Caption = "SPPVM Module"
    Height = 1095
    Left = 1680
    TabIndex = 0
    Top = 2880
    Width = 5175

Begin VB.CommandButton Command1
    Caption = "Shell Programs 1 and 2"
    Height = 375
    Left = 240
    TabIndex = 1
    Top = 480
    Width = 4455
End

End

Begin VB.Label Label3
    Caption = "By: M. E. Ekpenyong, N. M. Umoh, E. E. Ekong."

BeginProperty Font
    Name = "MS Sans Serif"
    Size = 12
    Charset = 0
    Weight = 400
    Underline = 0 'False
    Italic = 0 'False
    Strikethrough = 0 'False
EndProperty
    Height = 375
The QSORT.BAS program is written and saved as a QBASIC file and is as presented below:

```
DECLARE SUB qsort (ll, rl, a())
COMMON a()
CLS
CLEAR , , 10000
DIM a(500)
```
FOR y = 1 TO 500
    a(y) = INT(RND * 1000) + 1
NEXT y

t1$ = TIME$

PRINT "START TIME: ";t1$
CALL qsort(1, 500, a(l))
t2$ = TIME$
PRINT "STOP TIME: ";t2$
END

SUB qsort(l, r, a(l))
i = l: j = r
x = a(INT((l + r) / 2))
DO WHILE i < x
    i = i + 1
END DO
DO WHILE x < a(j)
    j = j - 1
END DO
IF i <= j THEN
    SWAP a(i), a(j): i = i + 1: j = j - 1
END IF
IF i < j THEN CALL qsort(l, j, a(l))
IF i < r THEN CALL qsort(i, r, a(l))
END SUB

The sequential version (QSORTSEQ.BAS) is as shown below:
DECLARE SUB qsort (ll, rl, a(l))
COMMON a(l)
CLS
CLEAR , , 10000
DIM a(1000)
FOR y = 1 TO 1000
    a(y) = INT(RND * 1000) + 1
NEXT y

t1$ = TIME$
PRINT "START TIME: ";t1$
CALL qsort(1, 1000, a(l))
t2$ = TIME$
PRINT "STOP TIME: ";t2$
END
SUB qsort(l, r, a(l))
    i = l: j = r
    x = a(INT((l + r) / 2))
    DO WHILE a(i) < x
        i = i + 1
    LOOP
    DO WHILE x < a(j)
        j = j - 1
    LOOP
    IF i <= j THEN
        SWAP a(i), a(j): i = i + 1: j = j - 1
    END IF
    IF i < j THEN CALL qsort(l, j, a(l))
    IF i < r THEN CALL qsort(i, r, a(l))
END SUB

Since the recursive version of quick sort is used for the illustration, a machine with large physical memory is required. This requirement is to avoid the out of stack error (when the array size is large).

The VB shell serves as a multi-platform for the execution of different programs. In the above module, the QSORT.BAS program shelled twice will execute on the VB Shell platform in the same memory space. This process drastically reduces the total runtime.

PERFORMANCE TIME ANALYSIS

Results obtained reveals the following:

<table>
<thead>
<tr>
<th></th>
<th>SPPVM</th>
<th>SEQ. VERS.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>First shell</td>
<td>Second shell</td>
</tr>
<tr>
<td>Time diff.</td>
<td>18 secs</td>
<td>13 secs</td>
</tr>
<tr>
<td>Speedup(Tseq/Tshell)</td>
<td>2.14</td>
<td>2.31</td>
</tr>
</tbody>
</table>

From the above result, the total time taken to run the SPPVM is \((15:14:32-15:14:09) = 23\) secs (without merging), which is less than \(57/2 = 28.5\) secs, i.e. \(1/n \times T_{seq}\). The efficiency \((e)\) of our module = \(T_{seq}/(T_{sppvm} \times N)\). From the result, the efficiency can be estimated to be approximately 0.99, if merge time was considered.

CONCLUSION

Developing SPPVM modules with Visual Basic allow programs to be shelled on a Multi-platform provided the Shell path is correctly specified. Thus it is advantageous for programmers to employ
these facilities to improve on the efficiency of their programs especially when executing large computational jobs.

REFERENCES

Chapell D., 1996. Understanding Active X and OLE. Microsoft press. Redmond, USA.


Davis, R. S., 1996. Learn Java Now. Microsoft Press. Redmond, USA