(25 March 2000)

Section 6 - Hardware Specifics

This section of the manual contains pages dealing in a general way with dynamic memory allocation in GAMESS, the BLAS routines, and vectorization.

The remaining portions of this section consist of specific suggestions for each type of machine. You should certainly read the section pertaining to your computer. It is probably a good idea to look at the rest of the machines as well, you may get some ideas! The directions for executing GAMESS are given, along with hints and other tidbits. Any known problems with older versions of compilers are described in the control language files themselves.

The currently supported machines are:

1) UNIX computers. This includes machines such as the IBM RS/6000, Compaq AXP, Sun SPARC, HP, Linux on Intel, and numerous others. Some of the others are parallel computers such as the IBM SP and Cray T3E.

2) IBM computers running any of the various MVS and VM operating systems (*IBM). VM is often called CMS. For IBM AIX systems, see category 1.

3) Compaq AXP or VAX computers under VMS (*VMS).

It is rather rare for anyone to run GAMESS on these latter two systems, so they are probably rather rusty. For example, there has been no provision made as yet for the Distributed Data Interface (DDI) routines to short-circuit themselves into a sequential version for these two.

Dynamic memory in GAMESS	6-2
BLAS routines	6-3
Vectorization of GAMESS	6-3
UNIX	6-4
IBM	6-8
VMS	6-10

dynamic memory in GAMESS

GAMESS allocates its working memory from one large pool of memory. This pool consists of a single large array, which is partitioned into smaller arrays as GAMESS needs storage. When GAMESS is done with a piece of memory, that memory is freed for other uses.

The units for memory are words, a term which GAMESS defines as the length used for floating point numbers (usually 64 bits, that is 8 bytes per word).

GAMESS contains two memory allocation schemes. For some systems, a primitive implementation allocates a large array of a FIXED SIZE in a common named /FMCOM/. This is termed the "static" implementation, and the parameter MEMORY= in $CONTRL cannot request an amount larger than chosen at compile time. Wherever possible, a "dynamic" allocation of the memory is done, so that MEMORY= can (in principle) request any amount. The memory management routines take care of the necessary details to fool the rest of the program into thinking the large memory pool exists in common /FMCOM/.

Computer systems which have a "static" memory allocation are IBM mainframes running VM or MVS, or Apollo and maybe a very few other Unix systems to which we have no direct access for testing purposes. If your job requires a larger amount of memory than is available, your only recourse is to recompile UNPORT.SRC after choosing a larger value for MEMSIZ in SETFM.

Computer which have "dynamic" memory allocation are VMS machines and almost all Unix systems. In principle, MEMORY= can request any amount you want to use, without recompiling. In practice, your operating system will impose some limitation. As outlined below, common sense imposes a lower limit than your operating system will.

By default, most systems allocate a moderate amount of memory: 750,000 words. This amount is adequate for almost all HF (RHF, UHF, ROHF, GVB) runs, although RUNTYP=HESSIAN may require more. Large GUGA runs (CI, MCSCF) may require an increased value for MEMORY in $CONTRL, perhaps to 2,000,000 words. EXETYP=CHECK runs will always tell you the amount of memory you need.

Many places in GAMESS implement an out of memory algorithm, whenever the in memory algorithm can require an excessive amount. The in memory algorithms will perform very poorly when the work arrays reside in virtual memory rather than physical memory. This excessive page faulting activity can be avoided by letting GAMESS choose its out of core algorithms. These are programmed such that large amounts of numbers are transferred to and from disk at the same time, as opposed to page faulting for just a few values in that page. So, pick an amount for MEMORY= that will reside in the physical memory of your system!

The object code and local storage for GAMESS compiles to about 5 Mbytes on most systems. Add this value to the number of Mbytes requested by MEMORY= (the conversion is multiply by 8, then divide by 1024 twice). For example, 750,000 words of memory leads to a total program size of 11 Mbytes. Depending on how many GAMESS jobs you run simultaneously, and the total number of Mbytes of physical memory installed in your system, you may be able to increase the MEMORY= value.

A general guideline is to select an amount of memory that will not be paged often. If your system has 64 Mbytes, and you are running only two copies of GAMESS at one time, a reasonable choice for MEMORY= would be to increase GAMESS to a total size of 28 Mbytes. That leaves some memory for the operating system.

The routines involved in memory allocation are VALFM, to determine the amount currently in use, GETFM to grab a block of memory, and RETFM to return it. Note that calls to RETFM must be in exactly inverse order of the calls to GETFM. SETFM is called once at the beginning of GAMESS to initialize, and BIGFM at the end prints a "high water mark" showing the maximum memory demand. GOTFM tells how much memory is not yet allocated.

BLAS routines

The BLAS routines (Basic Linear Algebra Subprograms) are designed to perform primitive vector operations, such as dot products, or vector scaling. They are often found implemented in assembler language in a system library, even on scalar machines. If this is the case, you should use the vendor's version!

The BLAS are a simple way to achieve BOTH moderate vectorization AND portability. The BLAS are easy to implement in FORTRAN, and are provided in the file BLAS.SRC in case your computer does not have these routines in a library.

The BLAS are defined in single and double precision, e.g. SDOT and DDOT. The very wonderful implementation of generic functions in FORTRAN 77 has not yet been extended to the BLAS. Accordingly, all BLAS calls in GAMESS use the double precision form, e.g. DDOT. The source code activator translates these double precision names to single precision, for machines such as Cray which run in single precision.

Machines which probably do provide assembler versions of the BLAS are all vector machines. They are also now frequently found on scalar machines. The compiling scripts use the system BLAS library whereever this is provided as a standard part of the operating system.

If you have a specialized BLAS library on your machine, for example because you purchased Compaq's DXML or Sun's Performance libraries, using them may give modest (a few percent) speedups. It is easy to use such a library:

a) remove the compilation of 'blas' from 'compall',
b) if the library includes level 3 BLAS, set the value of 'BLAS3' to true in 'comp',
c) in 'lked', set the value of BLAS to a blank, and set libraries appropriately, e.g. to '-ldxml'.

The reference for the level 1 BLAS is

C.L.Lawson, R.J.Hanson, D.R.Kincaid, F.T.Krogh ACM Trans. on Math. Software 5, 308-323(1979)

Vectorization of GAMESS

As a result of a Joint Study Agreement between IBM and NDSU, GAMESS has been tuned for the IBM 3090 vector facility (VF), together with its high performance vector library known as the ESSL. This vectorization work took place from March to September of 1988, and resulted in a program which is significantly faster in scalar mode, as well as one which can take advantage (at least to some extent) of a vector processor's capabilities. Since our move to ISU we no longer have access to IBM mainframes, but support for the VF, as well as MVS and VM remains embedded within GAMESS. Several other types of vector computers are supported as well.

Anyone who is using a current version of the program, even on scalar machines, owes IBM their thanks both for NDSU's having had access to a VF, and the programming time to do code

improvements in the second phase of the JSA, from late 1988 to the end of 1990.

Some of the vectorization consisted of rewriting loops in the most time consuming routines, so that a vectorizing compiler could perform automatic vectorization on these loops. This was done without directives, and so any vectorizing compiler should be able to recognize the same loops.

In cases where your compiler allows you to separate scalar optimization from vectorization, you should choose not to vectorize the following sections: INT2A, GRD2A, GRD2B, and GUGEM. These sections have many very small loops, that will run faster in scalar mode. The remaining files will benefit, or at least not suffer from automatic compiler vectorization.

The highest level of performance, obtained by vectorization at the matrix level (as opposed to the vector level operations represented by the BLAS) is contained in the file VECTOR.SRC. This file contains replacements for the scalar versions of routines by the same names that are contained in the other source code modules. VECTOR should be loaded after the object code from GAMESS.SRC, but before the object code in all the other files, so that the vector versions from VECTOR are the ones used.

Most of the routines in VECTOR consist of calls to vendor specific libraries for very fast matrix operations, such as IBM's Engineering and Scientific Subroutine Library (ESSL). Look at the top of VECTOR.SRC to see what vector computers are supported currently.

If you are trying to bring GAMESS up on some other vector machine, do not start with VECTOR. The remaining files (excepting BLAS, which are probably in a system library) represent a complete, working version of GAMESS. Once you have verified that all the regular code is running correctly, then you can adapt VECTOR to your machine for the maximum possible performance.

Vector mode SCF runs in GAMESS on the IBM 3090 will proceed at about 90 percent of the scalar speed on these machines. Runs which compute an energy gradient may proceed slightly faster than this. MCSCF and CI runs which are dominated by the integral transformation step will run much better in vector mode, as the transformation step itself will run in about 1/4 time the scalar time on the IBM 3090 (this is near the theoretical capability of the 3090's VF). However, this is not the only time consuming step in an MCSCF run, so a more realistic expectation is for MCSCF runs to proceed at 0.3-0.6 times the scalar run. If very large CSF expansions are used (say 20,000 on up), however, the main bottleneck is the CI diagonalization and there will be negligible speedup in vector mode. Several stages in an analytic hessian calculation benefit significantly from vector processing.

A more quantitative assessment of this can be reached from the following CPU times obtained on a IBM 3090-200E, with and without use of its vector facility:

	ROHF grad	RHF E	RHF hess	MCSCF E
scalar	168 (1)	164 (1)	917 (1)	903 (1)
vector	146 (0.87)	143 (0.87)	513 (0.56)	517 (0.57)

UNIX

GAMESS will run on many kinds of UNIX computers. These systems runs the gamut from very BSD-like systems to very ATT-like systems, and even AIX. Our experience has been that all of these UNIX systems differ from each other. So, putting aside all the hype about "open

systems", we divide the Unix world into four classes:

Supported: the Compaq AXP, HP 9000, IBM RS/6000, IBM SP, Intel Pentium under RedHat Linux, Sun ultraSPARC. These are the only types of computer we currently have at ISU, so these are the only systems we can be reasonably sure will work (at least on the models and release of O/S we are using). Both the source code and C-shell control language is correct for these.

Acquainted: ConvexSPP, Cray PVP, Cray SV1, Cray T3E, Fujitsu AP and VPP, Hitachi SR, NEC SX, and SGI. We do not own any of these systems at ISU, and so we cannot guarantee that these work. GAMESS has been run on each of these, but perhaps not recently. The source code for these systems is probably correct, but the control language may not be. Be sure to run all the test cases to verify that the current GAMESS still works on these brands.

Jettisoned: Alliant, Apollo, Ardent, Celerity, Compaq stations, FPS model 500, IBM AIX mainframes, Intel Paragon, Kendall Square, MIPS, NCube, Thinking Machines. In most cases the company is out of business, or the number of machines in use has dropped to near zero. Accordingly, these versions were dropped from the source code UNPORT.SRC and compiling scripts in fall 1997. In addition, support for Convex C-series and Cray T3D was removed in fall 1998. Of these, only the Celerity version's passing should be mourned, as this was the original UNIX port of GAMESS, back in July 1986.

Terra Incognita: everything else! You will have to decide on the bit packing, the contents of UNPORT, write the control language, and generally use your head.

You should have a file called "readme.unix" at hand before you start to compile GAMESS. These directions should be followed carefully. Before you start, read the notes on your system below, and read the compiler clause for your system in 'comp', as notes about problems with certain compiler versions are kept there.

Execution is by means of the 'rungms' script, and you can read a great deal more about its DDIKICK command, and scalability of parallel runs in the Programmer's Reference chapter of this manual.

Compaq AXP: These are scalar systems, running Tru64. At least, those are the current 2000 names. This category includes all AXP machines labeled Digital or Compaq on the front, and whose O/S is called OSF1, Digital Unix, or Tru64. The compiling script will invoke the f77 compiler, so read 'comp' if you have the f90 compiler instead. This version was changed to use native 64 bit integers in fall 1998. These machines run in parallel using the system's TCP/IP socket library.

You can also run GAMESS on AXP Linux, by using the Tru64 Compaq compilers, which permit the Tru64 version to run. Do not use g77 which allocates 32 bit integers, as the system's malloc routine for dynamic memory allocation returns 64 bit addresses, which simply cannot be stored in 32 bit integers. The Compaq compilers can easily generate 64 bit integers, so obtain FORTRAN and C from http://www.unix.digital.com/linux/software.htm We tested cfal-1.0-7 and ccc-6.29.002-2. Make the following modifications by hand before compiling:

1. vi compall, to change "cc" to "ccc" in three places
2. vi comp, changing "f77" to "fort" in compaq-axp clause
3. vi lked, changing "f77" to "fort" in compaq-axp clause

4. vi source/ddisoc.c, to add a second trailing underscore to all "soc" routine names in the #ifdef COMPAQ clause.

Then compile and link using target 'compaq-axp'.

ConvexSPP: the SPP is a parallel system built with HP PA-RISC nodes, and so this version is derived from the HP version. It differs only in the control language used to compile and link, which was supplied to us by Dave Mullally of Convex Corporation. Be careful checking the test inputs on this system. (system uses MPI to run parallel?)

Cray PVP: this means the C90 type vector systems. Thanks to Dick Hilderbrandt, Kim Baldridge, Richard Walsh, and Howard Pritchard for their help with Crays and UNICOS. This is a 64 bit system, so be sure to activate the "*SNG" line in actvte.code, in addition to "*UNX". This version should be reasonably reliable, but we don't run on Cray vector systems often anymore, and the parallelization needs a rethink (e.g. MPI should probably be used).

Cray SV1: this is a parallel and vector system from Cray, for which Charles Castevens of SGI has provided a port in March 2000. The present version uses the socket code to run in parallel, as the present version of the MPI library for this machine does not run all the examples.

Cray T3E: A massively parallel computer from Cray. This machine uses its native SHMEM library for effective use of distributed data (and all other messages too). We use this version on a DoD T3E regularly, and it should install and run quite well. We thank Howard Pritchard for his help with understanding SHMEM.

Digital: See Compaq above.

Fujitsu: The AP is a parallel system (SPARC nodes), and the VPP is a vector system. The *FUJ code in UNPORT and VECTOR was written by Ross Nobes in August 1991 for the VPP line. The VPP version was verified in February 1998 by Anthony Russell, and the AP version added by Ajay Limaye at the same time, in Alistair Rendell's group at Australian National University. Both machines have MPI-1 and this should be used to run in parallel, but these systems need to have their control language brought up to date to support DDI.

Hitachi SR2201. This is a MPP platform. This version is due to Masato Yamanishi of Tokyo University. The MPI library on this system is used to run in parallel. However, hese systems need to have their control language brought up to date to support DDI.

HP: Any PA-RISC series workstation. The HP-UX port is due to Fred Senese, Don Phillips, and Tsuneo Hirano. Dave Mullally at HP has kept this version going since, and in 1998, sited a HP loaner system at ISU. As a consequence, this version is now carefully checked. The compiling script selects the most recent instruction set, and will need to be changed to +DA1.1 if you have an older model PA-RISC chip. These workstations run in parallel using the system's TCP/IP socket library.

IBM: "superscalar" RS/6000. There are two targets for IBM workstations, namely "ibm32" and "ibm64". The latter should be used only in the following conditions:

a) you have a Power3 based machine, which is not a SP.
b) you are running AIX 4.3.1 or higher
c) you have XL FORTRAN 5.1.1. or higher
d) you have more than 2 GBytes of memory installed.

This target uses a slow plain-vanilla FORTRAN matrix multiplication routine instead of a tuned library routine (since IBM do not provide a fully 64 bit ESSL), so it should be used only if you need access to a large memory. All other situations should compile with "ibm32". If you use 32 bit compilation on a Power3, you should probably change the 'comp' script so that ARCH and

TUNE equal "pwr3", but leave these unchanged for every other chip IBM has sold. The compiling script will activate *IBM lines everywhere, except *UNX in IOLIB and *AIX in UNPORT. Parallelization is achieved using the TCP/IP socket calls found in AIX.

IBM-SP: The SP parallel systems, based on RS/6000 workstations. Parallelization is accomplished using the MPI-1 library found in IBM's POE software. To be quite frank, the current version of GAMESS does not run nearly as well as it should on old IBM SP systems. "Old" means possession of an old switch ("lscfg -v -l css0" reports High Performance Switch Type 2, new replies "SP Switch") and use of old software ("lslpp -h ppe.poe" reports 2.3 or older). Unfortunatly our system at ISU suffers from both maladies! If you have an old system, ask Mike for a replacement for DDI.SRC that cuts out the parallel MP2 program, but lets everything else run well. If you have a newer system, ask Mike for control language to run properly under LoadLeveler and POE's recent versions. Note that since IBM does not provide a 64 bit MPI library, use of 64 bits to address memory is completely impossible.

Linux-PC: this means Linux Intel systems, using f2c and gcc as a "FORTRAN compiler". This version was written by Pedro Vazquez in Brazil in December 1993, and modified by Klaus-Peter Gulden in Germany. The usefulness of this version has matched the growth of interest in PC Unix, due to the improvement in Intel CPU performance, and increases in memory and disk capacities, to near workstation levels. We acquired a 266 MHz Pentium-II PC running RedHat Linux in August 1997, and founnd it performed flawlessly. In 1998 we obtained six 400 MHz Pentium-IIs for sequential use, and in 1999 a 16 PC cluster, running in parallel day in and day out. Parallelization is accomplished using Linux's TCP/IP socket library to support DDI. We do not support Linux on AXP at present for technical reasons. Note that we use only RedHat Linux, and from time to time hear that there are glitches with other releases. Since RedHat 6.0 omits the f2c interpreter, there is a g77 compiling clause commented out in "comp" for you to try...beware, we do not use this in our own day to day production runs yet.

If you have the commercial pgf77 compiler, some tips from Fred Arnold are:

(a) in comp: pgf77 -c -tp p6 -fast $MODULE.f
(b) in ddisoc.c, use only one underscore, as indicated.
(c) presuming that you compiled the .c modules with gcc, lked needs to have "-g77libs" added to LDOPTS.

He says this is about 10% faster than g77.

NEC SX: vector system. This port was done by Janet Fredin at the NEC Systems Laboratory in Texas in 1993, and she periodically updates this version, including parallel usage, most recently in August 1999. You should select both *UNX and *SNG when manually activating ACTVTE.CODE, and compile actvte by "f90 -float0 -w -o actvte.x actvte.f". This version uses *CRY lines, except *UNX in IOLIB and *NEC in UNPORT and VECTOR. Parallelization is achieved using MPI-1 messaging calls.

Silicon Graphics: scalar system. This version is used by a fair number of people, so the source code is reliable. SGI has sold machines with R2000, R3000, R4x00, R5000, R8000, R10000, and R12000 processors. If you are not sure which you have, type "hinv" to find out. The target "sgi-mpi" assumes you have a large Origin 2000 class machine, on which the MPI library has been installed. The target "sgi" assumes you have a R4x00 chip, and will be using sockets. Since SGI has marketed a lot of machines between these two limits, you may want to use compiler flags more like the "sgi-mpi" target in the "sgi" parts of the compiling scripts. See 'comp' for more information on this. If you use the target "sgi-mpi" your system needs to have had the MPI-1 library installed on it. This library part of the "message passing toolkit" which is offered free by SGI, at

http://www.sgi.com/Products/Evaluation

We thank Omar Stradella of SGI for help with the MPI version on large SGI systems. The

compiling target "sgi" is tested much more often by us than "sgi-mpi".

Sun: scalar system. This version is tuned for the new ultraSPARC 2 chips, running Solaris. If you have an older SPARC system, change the 'comp' script's choice for the architecture. Solaris 2.6 can write FORTRAN disk files larger than 2 GBytes, as we have verified, if you download the replacement I/O library from

http://www.sun.com/workshop/performance/largefiles.html

Presumably Solaris 7 incorporated this into itself. The parallelization is accomplished by using Solaris' TCP/IP socket library to support DDI. Since Sun has located an E450 model at ISU, this version is very reliable.

IBM

GAMESS runs on all IBM S/370 equipment such as the 438x, 308x, and 3090 systems (and plug compatibles) under any of the MVS, MVS/XA, MVS/ESA, VM, VM HPO, or VM/XA systems, as all of these support the exact same compiler, VS FORTRAN. IBM AIX systems are described with the other UNIX systems.

We do not have access to IBM mainframes at Iowa State, so the VM, MVS, and AIX/370 versions have not been tested by us since summer 1992. However, the IBM version has been in use for so long prior to this that it should still be reliable. 1992 was a long time ago, so good luck...

XA: The source code assumes that you have one of the extended address systems, by which we mean either XA or the newer ESA. This means the file UNPORT.SRC has a dimension of 5,000,000 for the /FMCOM/ dynamic memory pool. If you have XA, use the compiler option DC(FMCOM) to put this common block above the line. If you do not have an XA system, then this dimension is way too large to fit "below the 16 Mbyte line". Before you compile, you must change the dimension of /FMCOM/, perhaps to 750,000 words, in order to obtain a 12 Mbyte load module. Of course, do not use the DC(FMCOM) compile option in this case. The control language provided assumes XA in *.MVS files, and assumes its absence in the *.CMS files. The *.CMS files have comments showing how to change things if you have VM/XA on your system.

In the following, MVS means either MVS, MVS/XA, or MVS/ESA, and likewise VM means VM, VM HPO, or VM/XA.

Vectorization: The control language and source code which we distribute assumes you have a scalar IBM system. If your system does have a VF attached, then follow the comments in the VM or MVS control language which show you how to change these to use a VF. (You can find all these places by a string search for "VF"). In addition, change UNPORT.SRC's subroutine ASKVEC so that it returns true for IBM equipment.

Assembler: The IBM version comes with two assembler routines for timing purposes. These have been tested with both the F and H level assemblers. They are provided instead of using CLOCK and DATIM in VS FORTRAN version 2 for backward compatibility with VS FORTRAN version 1. They also work the same under VM and MVS, which is nice.

Compiler: The FORTRAN compiler which we last tested is VS FORTRAN 2.4.0. You should use OPT(3) and NOSDUMP for all modules. We used VS FORTRAN 1.4.1 for several years, this is a very reliable compiler. Some versions of 2.3.0 will not vectorize SGRAD and SHESS, in GRD1 and HSS1B respectively. The symptom is a bombout at compile time, the fix is to use PARM=NOVECTOR.

For some reason, the VS FORTRAN compiler does not accept the FORTRAN 77 syntax "COMPLEX*16 FUNCTION XXX" used in two places in ZHEEV.SRC. You should change these to read "COMPLEX FUNCTION XXX*16" before you compile.

VM: The distribution contains some EXEC's named *.CMS (you should COPYFILE * CMS B = EXEC =) for both compilation and execution. The COMPALL EXEC does not require ACTVTE, instead it uses XEDIT. These EXEC's assume that the VMBATCH software has been installed on your system, but they ought to be easy to modify for other batch facilities. A 20 cylinder (3380) minidisk is capable of storing both the source code (in V record format, to avoid storing trailing blanks), and the object code TEXTLIB. This minidisk must be linked in read-only mode by VMBATCH when GAMESS is run. Any co-workers can also link in read-only mode, so that only one copy of the EXEC for execution and the TEXTLIB is needed. Thus, the GAMESS minidisk probably should not be your main A-disk. You must specify the number of extents for file DASORT. You may want to experiment with creating a single record direct file, with the correct file lengths with a small FORTRAN program. COPYFILE this single record file to your job's DASORT file, and this file will not be filled with zeros when first open, and it will grow only to the size it needs to be.

MVS: The *.MVS files are JCL control language for this operating system. The *.MVS files are correct, to the best of my knowledge. However, they have not been used in a long time, and mistakes can easily creep into JCL! The MVS system provides a very nice feature that lets disk files span more than one physical volume. Look at the control language for AOINTS for an example.

VMS

The VMS version of GAMESS is supposed to be correct for VAX or Alpha based VMS systems. It has been a very long time since we tested a VMS system, however. We did use VAX systems for over a decade, but have not been able to do so since about 1993. Be careful running the test cases on VMS! The graphics programs will run properly under DECwindows, and can also produce PostScript output.

Both VMS versions are identical, with one exception. The line calling ERRSET in BEGING in UNPORT.SRC cannot be used on an Alpha, so it is commented out CVAX. You should change this line with a text editor to *VMS if you are on a VAX based system. More detailed compiling instructions can be found in the README.VMS file.

Your environment for running GAMESS should include four logical variables, namely

```
$ ASSIGN DKA300:[MIKE.GAMESS]           GAMESS:
$ ASSIGN DKA300:[MIKE.GAMESS.TOOLS]     TOOLS:
$ ASSIGN DKA300:[MIKE.GAMESS.GRAPHICS]  PLT:
$ ASSIGN DKB500:[SCRATCH.MIKE]          SCR:
```

The latter is any disk area where you can write very large scratch files while the batch jobs are running. It is best if everyone has their own separate directory in this scratch area. If your system manager has not already made the above logical assignments for you, you can place these in your LOGIN.COM file.

System managers should note that if you have more than one disk, you can use MOUNT/BIND to achieve as large as possible a "logical disk" for SCR:. Users may need to have their WSEXTENT and PGFLQUO values in AUTHORIZE increased to allow "reasonable" physical and virtual memory use (16 MB and 24 MB respectively??? although "reasonable" values will depend on your installed memory). These AUTHORIZE values may require adjustment of SYSGEN parameters WSMAX and VIRTUALPAGECNT.