(10 November 2005)

```
**********************************
*                                *
* 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 a good idea to look at the rest of the machines as well, as you may get some ideas! The directions for executing GAMESS are given, along with hints and other tidbits. Any known problems with certain compiler versions are described in the control language files themselves, not here.

The currently supported machines are all running Unix. The embedded versions for IBM mainframes and VAX/VMS have not been used in many years, and are no longer described here. There are binary versions for Windows available on our web site, but we do not supply a source code version for Windows (except that the Unix code will compile under the Cygwin Unix environment for Windows). Please note that with the OS X system, the Macintosh is considered to be a system running Unix, and is therefore well supported.

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, 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 MWORDS in $SYSTEM cannot request an amount larger than chosen at compile time. Wherever possible, a "dynamic" allocation of the memory is done, so that MWORDS 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 "static" memory allocation are IBM mainframes running VM or MVS 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 all Unix systems and VMS. In principle, MWORDS 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 small amount of memory: one million words. This amount is quite small by modern standards, and therefore exists on all machines. It is left up to you to increase this with your MWORDS input to what your machine has. EXETYP=CHECK runs will always tell you the amount of memory you need.

Many computations in GAMESS implement out of memory algorithms, 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 MWORDS that will reside in the physical memory of your system! MWORDS, multiplied by 8, is roughly the number of Mbytes and should not exceed more than about 90% of your installed memory (less if you are sharing the computer with other jobs!).

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 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.

If you have a specialized BLAS library on your machine, for example IBM's ESSL, Compaq's CXML, or Sun's Performance Library, using them can produce significant speedups in correlated calculations. The compiling scripts attempt to detect your library, but if they fail to do so, it is easy to use one:

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 '-lessl'.

Check the compilation log for mthlib.src, in particular, to be sure that your library is being found. It has a profound effect on the speed of MP2 and CC computations!

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)

Notes for specific machines

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: Apple MAC under OS X, HP/Compaq/DEC AXP, HP PA-RISC, IBM RS/6000, Intel Pentium and AMD 32 bit equivalents, 64 bit Intel Itanium2, and 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 hardware model and O/S release we are using). Both the source code and control language is correct for these.

Acquainted: 64 bit AMD chips (e.g. Opteron), Compaq SuperCluster, Cray PVP, Cray X1, Cray XD1, Fujitsu PP, IBM SP, NEC SX, and SGI MIPS. We don't have any of these systems at ISU, so we can't 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, Convex, Cray T3D, DECstations, FPS model 500, Fujitsu AP and VPP, HP Exemplar, Hitachi SR, IBM AIX mainframes, Intel Paragon, Kendall Square, MIPS, NCube, and Thinking Machines. In most cases the company is out of business, or the number of machines in use has dropped to near zero. 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 contents of UNPORT, write the scripts, 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 in the installation guide 'readme.ddi'. Note in particular that execution of GAMESS now uses System V shared memory on many systems, and this will often require reconfiguring the system's limits on shared memory and semaphores, along with a reboot. Full details of this are in 'readme.ddi.

Users may find examples of the scalability of parallel runs in the Programmer's Reference chapter of this manual.

* * * * * *

AMD Opteron (and other AMD 64 bit chips): These are assumed to be running Linux, and to have the Portland Group pgf77 compiler installed. The port was done by Shiro Koseki and Hiroaki Umeda in July 2004, with some input from Ted Packwood. Note that unlike the 32 bit lines, the 64 bit chips from AMD and Intel are different, using different compilers. At present, the various Linux projects "g90" and "gfortran" seem incapable of producing correct GAMESS compilation, so we recommend the Portland Group Compiler.

Compaq AXP: These are scalar systems. This category means any AXP machines, whether labeled Digital or Compaq or HP on the front, with an O/S called OSF1, Digital Unix, or Tru64. It also includes systems running Linux, see below. The unique identifier is therefore the AXP chip, "alpha".

High end Compaq systems such as the SuperCluster product support a SHMEM implementation for one-sided message passing, use target 'compaq-shmem' for this.

The compiling script invokes 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.

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://h18000.www1.hp.com/products/software/alpha-tools
Then compile and link using target 'compaq-axp'.

Cray XT3: a massively parallel platform, based on dual Opteron processor blades connected by Cray's 3D mesh, running a microkernel named Catamount. The SHMEM library is not ready yet, and the microkernel does not permit the use of TCP/IP, so the parallel support is the MPI code in the original DDI implementation. Porting to this machine was done by Ted Packwood of Cray in October 2005.

Cray XD1: This is a system composed of 2-way SMP blades, based on Opteron chips, and using the Rapid Array custom network. This version uses TCP/IP as the message passing layer. The port to this machine was supplied by Ted Packwood of Cray in June 2005.

Cray X1 and X1E: this is a vector processor system, using the SHMEM library as its parallel support. This version was supplied by Ted Packwood of Cray in March 2003.

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.

Cray PVP: this means J90 and SV1 type vector systems. Thanks to Dick Hilderbrandt, Kim Baldridge, Richard Walsh, and Howard Pritchard for their help with Cray systems and UNICOS. This is an obsolescent product line, and this version has not been tested in some time. TCP/IP sockets and System V memory are the parallel scheme used.

Digital: See Compaq above.

Fujitsu: The PrimePower is a parallel system based on SPARC CPUs, and so is much like a Sun, although it uses different compilers. The control language for this was written by Roger Amos at Australian National University Supercomputer Facility in March 2003 and updated in January 2005, with Dmitri Fedorov contributing changes for the new DDI code in 2004. If your system has less than 2 Gbytes of memory per processor, request the 32 bit version, as this runs a bit (5%) faster than the 64 bit version. The latter should be used in order to address a larger installed memory.

HP: Any Itanium2 or PA-RISC series workstation. Help with this version has come from due to Fred Senese, Don Phillips, Tsuneo Hirano, and Zugmunt Krawczyk. Dave Mullally at HP has been involved in siting HP systems at ISU, presently Itanium2 based. So, we used 'hpux32' for many years, but are now running only the 'hpux64' version. The latter version can be considered to be carefully checked since it is in use at ISU, but please be a little more careful checking tests if you try 'hpux32'.

IBM: "superscalar" RS/6000. There are two targets for IBM workstations, namely "ibm32" and "ibm64", neither of these should be used on a SP system. The 64 bit version should be used only in the following conditions:

a) you have a Power3 machine, or newer
b) you are running AIX 4.3.1, or higher
c) you have XL FORTRAN 5.1.1, or higher

All other situations should compile with "ibm32". In case you have only Power4 or Power5 chips, consider changing the -qtune and -qarch options from pwr3 to match your case. Parallelization is achieved using the TCP/IP socket calls found in AIX.

IBM-SP: The SP parallel systems. This is a 64 bit implementation. The new DDI library will operate with LAPI support for one-sided messaging, and a special execution script for LoadLeveler is included.

IBM Blue Gene: This is a massively parallel machine, made with the 32 bit PowerPC 440 chip with a dual FPU, with a fixed memory of 512 MB RAM. The parallelization uses the original DDI version, running in pure MPI mode. Execution on this machine is a bit peculiar, so there is a special script misc/bggms, used instead of the normal 'rungms'. The Blue Gene port was done by Brian Smith of IBM and Brett Bode, and included in GAMESS in June 2005.

Linux-ia64: this means the Intel Itanium family of processors, running under 64-bit modified versions of the RedHat operating system. Such systems are available from a number of vendors, including the SGI Altix, NEC T7, HP, and others. The compilers are Intel's ifc for FORTRAN and icc for C. Intel's MKL library for the BLAS can be linked for extra performance. Please see more in the 'comp' script about obtaining these compilers, and their version numbers. Fred Arnold of Northwestern University created the initial port, in June 2002. Additional assistance with this version came from NEC's Jan Fredin, including use of NEC's MathKeisan library for the BLAS. Since June 2003, we have

had a SGI Altix on loan at Iowa State University, so this version is now well tested as a full 64 bit program. This version relies on TCP/IP sockets and System V memory for message passing. Compile the source activator by "ifc -o actvte.x -Vaxlib actvte.f". Be sure to use the MKL library only in serial mode, with the MKL_SERIAL environment value set to YES.

Linux-PC: this means Intel/Athlon 32 bit chips. This version is originally due to Pedro Vazquez in Brazil in 1993, and modified by Klaus-Peter Gulden in Germany. The usefulness of this version has matched the steady growth of interest in PC Unix, due to the improvement in CPU, memory, and disks, to workstation levels. We acquired a 266 MHz Pentium-II PC running RedHat Linux in August 1997, and found 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. At present we have a number of 32 bit Athlons, running Fedora Core version 1. We have used RedHat 4.2, 5.1, 6.1, and 7.1 prior to Core 1, and therefore feel that essentially any RedHat distribution should work.

Parallelization is accomplished using Linux's TCP/IP socket library, and System V memory to support the DDI operations. We discourage the use of MPICH under Linux, since the use of TCP/IP is efficient, and installation will be trouble free.

Compilation defaults to g77, but you can select f2c/gcc or the commercial products ifc from Intel, or pgf77 from Portland Group. See 'comp' for more information about compilers.

Downloading a BLAS library from the network for use on LINUX is heartily recommended. Some URL's are

```
  http://www.cs.utk.edu/~ghenry/distrib/index.htm   <- ASCI
  http://sourceforge.net/projects/math-atlas
  http://developer.intel.com/software/products/mkl
  http://www.cs.utexas.edu/users/flame/ITXGEMM
```

We are using the ASCI 1.2F precompiled Pentium2 library for both Pentium and Athlon processors. If your library isn't named /usr/local/bin/libblas-asci.a, simply change the "comp" and the "lked" scripts to match your name choice.

Macintosh OS X: This is for Mac running OS X, which is a genuine Unix system "under the hood". This version closely resembles the Linux version for PC, including 2 GB file size limitations. The compiler is g77, but check out

the notes in 'comp' about xlf if you have a G5. Notes in 'comp' will tell you how to get these compilers. TCP/IP sockets and System V memory are used as the message passing transport for parallel jobs. Added bonus: A Macintosh will be capable of running the MacMolPlt visualization program.

Here are two tricks for Tiger, using the mac32 target, which we hope to make invisible someday. But right now, the scripts are presently correct for Panther, and Tiger requires two hand edits:

a) in 'compddi', remove the -Dsocklen_t=int parameter

b) in 'compall', add -DCLK_TCK=100 to "extraflags"

Of course, both of these apply to the 'mac32' sections. This has been tested with gcc 4.0 from Apple, and g77 3.4 from Fink.

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 Oct. 2003. You should select both *UNX and *SNG when manually activating ACTVTE.CODE, and compile actvte by "f90 -ew -o actvte.x actvte.f".

Silicon Graphics: This refers to MIPS based systems. For SGI's Itanium2 based systems marketed as "Altix", see the 'linux-ia64 section'. This version is used at a fair number of external sites, so the source code is reliable. SGI has sold machines with R2000, R3000, R4x00, R5000, R8000, R10000, R12000, and R14000 processors. You can type "hinv" to find out which you have. The targets "sgi32" and "sgi64"assume that you have the newest processor, if not, change the architecture from '-r12000' and instruction set from '-64 -mips4' if you have older equipment. (For example, the instruction set might be '-mips2' on very old machines, or '-n32 -mips3'). For many years, the compiler optimization was set at -O2 to try to avoid numerical problems, but in 2002 it was reset to -O3. If you find the test examples don't work, please recompile with the safer level -O2 optimization.

The SGI version uses System V shared memory now. Do not attempt to use MPI, or the web patches from Omar Stradella that you might be familiar with from previous versions of GAMESS on Origin.

Sun: scalar system. This version is set up for the ultraSPARC chips, running Solaris. The 64 bit version will run faster (due to using the SPARC v9 rather than the v8 instruction set), so most people should select the "sun64"

compiling target. If you have an older SPARC system, or if you are running Solaris 2.6, modify the 'comp' script as necessary. Install the SunPerf library from the compiler suite for maximum BLAS performance. Parallelization is accomplished using TCP/IP sockets. Since Sun located an E450 system at ISU in 1998, and two Sunfire 280R systems in 2002, this version is very reliable.