(6 May 2010)

```
**********************************
*                                *
* 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, 64 bit Intel/AMD chips such as the Xeon/Opteron/Itanium, 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: Cray XT, IBM SP, SGI Altix/ICE, 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 offsite, perhaps recently, but perhaps not. 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 T3E, Cray vectors, 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 death 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 chips: see "linux64" below.

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, so the compiling target is 'axp64', rather than a company name.

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 XT: a massively parallel platform, based on dual Opteron processor blades connected by Cray's 3D mesh, running a node O/S called "Compute Node Linux". The message passing involves a DDI running over MPI with a user selectable number of data servers. Unfortunately, the DDI is not fully integrated into our main code yet, and the scripting is a bit rough. Good support for these (XT3 through XT6) is expected by summer 2010.

Digital: See AXP above.

HP: Any Intel or PA-RISC series workstation or server. Help with this version has come from due to Fred Senese, Don Phillips, Tsuneo Hirano, and Zygmunt 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 use '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. 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 target is "ibm-bg". The older BG/L has been outmoded by the BG/P, but we still have an "L" at ISU. These are massively parallel machine, using a 32 bit PowerPC, and a limited amount of node memory. The "L" uses DDI running over the ARMCI library, running in turn over MPI, so the "L" does not use data servers. The "P" uses a straightforward DDI to MPI interface, with data servers. The "L" port was done by Brian Smith of IBM and Brett Bode at ISU, included in GAMESS in June 2005, and changed to use ARMCI in 2007 by Andrey Asadchev of ISU. Nick Nystrom's initial port to the "P" system was polished up by Graham Fletcher at Argonne National Labs in 2010. Special notes, and various files to be used on this system are stored in the directory ~/gamess/machines/ibm-bg.

Linux32: this means any kind of 32 bit chips, but typically is used only when "uname -p" replies "x86". Nearly every other chip is 64 bits, so see also Linux64 just below. 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. We have used RedHat 4.2, 5.1, 6.1, 7.1, and Fedora Core 1, prior to switching over exclusively to 64-bit Linux. This version is based on gfortran or g77, gcc, and the gcclib, so it should work for any kind of 32 bit Linux. This version uses 'sockets' for its message passing. The configuration script will suggest possible math library choices to you.

By 2010, probably most Linux systems in existence are 64-bit capable, so the next version is more better!

Linux64: this means any sort of 64 bit chip running an appropriate 64 bit Linux operating system. The most common "linux64" build is on AMD or Intel chips, where "uname -p" returns x86_64 or ia64. However, if you choose the 'gfortran' compiler, no processor-specific compiler flags are chosen, so this version should run on any 64-bit Linux system, e.g. AXP or SPARC.

If you are running on Intel/AMD processors, the configuration script lets you choose various FORTRAN compilers: GNU's gfortran, Intel's ifort, Portland Group's pgf77, and Pathscale's pathf90. You can choose a variety of math libraries, such as Intel's MKL, AMD's ACML, or ATLAS. You can choose to use MPI if your machine has a good network for parallel computing, with the options for the MPI type specified in detail in the file, but sockets are an easy to use alternative to MPI.

The choices for FORTRAN, math library, and MPI library can all be "mixed and matched". Except for 'gfortran', almost all this software has to be added to a standard Linux distribution. It is your responsibility to install what you want to use, to set up execution paths, to set up run time library paths (LD_LIBRARY_PATH), and so forth. The 'config' script will need to ask where these software packages are installed, since your system manager may have placed them almost anywhere.

Macintosh OS X: This is for Apple running OS X, which is a genuine Unix system "under the hood". This version closely resembles the Linux version. Installation of Apple's XCODE (from the OS X distribution DVD) gives you a C compiler and a math library. You can obtain a FORTRAN compiler (gfortran for 64 bit or g77 for 32 bits) from the wonderful web site of Gourav Khanna:

http://hpc.sourceforge.net

Request target "mac32" if your OS X is 10.4, or "mac64" if your OS X is 10.5 or newer.

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: The modern product line of this company is called Altix or Altix ICE. The operating system is Linux, the chips are 64 bit Intel processors, and the natural compiler and math library choices are Intel's ifort and MKL. The SGI software ProPack turns these commodity components into a supercomputer, and DDI should use the MPI library 'mpt' found in ProPack. Accordingly, the compiling target should be 'linux64', selecting ifort, MKL, and then mpt.

Silicon Graphics: The ancient product line of this company use various MIPS chips such as R4x00, R5000, R12000, etc. There are very few of these machines left, so target's 'sgi32' and 'sgi64' should be regarded as "rusty". The 32 bit target uses sockets communications, while the 64 bit one will use an old SHMEM interface, that only partially implements DDI. Hence FMO and parallel CCSD(T) will not run on 'sgi64'.

Sun: scalar system. This version is set up for the ultraSPARC or Opteron chips, running Solaris. The target for either chip is "sun64" as the scripts can automatically detect which one you are using, and adjust for that. Since Sun provided a ultraSPARC E450 system in 1998, two ultraSPARC3 Sunfire 280R systems in 2002, and a Opteron V40Z system in 2006, to the group at Iowa State, the Sun version is very reliable. Install the SunPerf math library from the compiler suite for maximum BLAS performance. Parallelization is accomplished using TCP/IP sockets and SystemV shared memory.