Report:
Running EGS4 on different architectures

A F Bielajew and S Walker†
Institute for National Measurement Standards
National Research Council of Canada
Ottawa, Canada
K1A 0R6
Tel: 613-993-2197
FAX: 613-952-9865
e-mail: alex@irs.phy.nrc.ca

†The Lawrence H. Lanzl Institute of Medical Physics
3600 - 15th Ave. W., Suite 205
Seattle, WA 98119
U S A
Tel: 206-286-0241
FAX: 206-286-0231
e-mail: suzan.walker@lanzl.com

National Research Council of Canada Report PIRS-0392
Naeser’s Law:
“You can make it foolproof, but you can’t make it damnfoolproof.”

1 Introduction

This lecture is a highly personalised description of various issues related to running EGS4 on different machines. The author’s direct experience comes from running EGS4 on the following architectures:

- VAX/VMS (e.g. 11/780 FPA, μVAX 3600, VAX 8xxx)
- IBM/CMS 3090/180/300/300E (HPO, VF)
- FPS-264 (M64/50)
- IBM PC 386/387 (OS386)
- RISC/UNIX machines (Dec’s, SGI’s, Sun’s, HP’s, IBM RS6000’s)

EGS4 has run on a variety of machines from PC/286’s to Cray’s as well as a variety of architectures, scalar, parallel and vector.

In this lecture I will attempt to address what problems one might expect in getting EGS4 running on a scalar (serial processor) machine. I leave detailed discussion of other architectures to a later date (when the author can get his hands on some).

2 Questions posed by EGS-perts and in-EGS-perts

2.1 What machine should I run EGS4 on?

Usually the response to this question is, “Whatever happens to be available at my institution!” In this case the proper question is, “Can I get EGS4 running on this machine?”.

The minimum configuration for EGS code is to have about 500KB of available main memory, about 20 MB of disk storage and a FORTRAN compiler. Increasing both main memory and disk capacities will increase one’s ability to handle complex problems. Monte Carlo simulations are generally compute-bound problems, so if economy of funds is important one does not need to invest in the fastest disk storage available. (It is nice to have, however.) Other desirable software/hardware/peopleware capabilities in approximate order of priority are:

- Floating-point processors, (e.g. VAX FPA’s, 80x87’s, Weitek x167’s) (almost a waste of time without these)
- Lots of memory. Fixed memory machines (e.g. DOS-PC’s, Cray’s) will not allow your code to run if it is too big. Virtual memory machines can slow down by a factor of 10^4 or more(!) if your job does not fit into memory and it employs disk space as virtual memory.
- Fortran debugging tools
- Graphics capabilities
- Someone else to run the system for you
2.2 How fast is fast?

There are many measures of machine speed, MIPS, MFLOPS, Whetstones, Dhrystones, Specmarks, Khornstones, LINPACK benchmarks, etc... Peak machine speed, peak-MIPS or peak-MFLOPS, (often quoted by manufacturers) or peak-anything is an almost useless measure of a machine's speed. The only possible interpretation of peak speed is, "The speed your application is guaranteed not to surpass". The most useful benchmark of a machine's speed is to time your own code running on it. To this end, we distribute a standard timing benchmark code [1] (called XYZDOS, part of the UNIX and PC distributions), for comparison of various machines and architectures. The next best estimate can be gleaned from the LINPACK benchmarks [2], based on solving a dense set of linear equations of order 100. This benchmark tests not only processing speed but also non-local and distant memory fetches, testing main memory–data cache–CPU bottlenecks, usually the cause of peak speed degradation. Monte Carlo calculations involve a lot of remote addressing and in this respect only are similar to linear algebra on large matrices. However, Monte Carlo codes usually involve iterative loops over large instruction sets and main memory–instruction cache bottlenecks can significantly affect performance. We have noted some anomalies between the comparison of XYZDOS and LINPACK.

Consider the comparison of an EGS4 simulation based on a realistic problem, the XYZDOS benchmark given in the following Table. The most remarkable revelation from the benchmark timing results is that high-end PC's (costing about $2K–$3K) and low-end UNIX workstations (costing about $3K–$6K) have encroached upon mainframe territory in this form of scientific computing.

If you are lucky enough to able to choose machines based upon EGS4 demands, choose the fastest, serial processing machines that return the highest benchmark numbers per unit of money. Note that apart from system maintenance, the number of machines should not be a strong factor in the determination. In the near future, network operating systems will be able to make good use of either many slower machines or a few fast ones. If you are unable to run your code or the XYZDOS benchmark, base your decision regarding machine speed by consulting the above list and seeing which architecture matches most closely, especially with respect to the size of the instruction and data caches. Failing this, use the LINPACK, Specmark, or MIPS timing benchmarks, but keep in mind that it may not be accurate. (The author has noted that for one architecture, the LINPACK benchmark and the XYZDOS benchmark differed by a factor of 5!)
Monte Carlo Timing comparisons last edited January 16, 1997

<table>
<thead>
<tr>
<th>MACHINE</th>
<th>O/S</th>
<th>FORTRAN</th>
<th>RATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX 11/780 FPA</td>
<td>VMS 3.7</td>
<td>FORT-11 3.7</td>
<td>≡ 1</td>
</tr>
<tr>
<td><strong>DOS/Windows/NT machines</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compaq 20 MHz 386/387</td>
<td>OS386 1.0</td>
<td>Lahey F77L-EM/32 2.0</td>
<td>1.11⁸⁸</td>
</tr>
<tr>
<td>ACAD 25 MHz 386/387</td>
<td>OS386 2.1.04</td>
<td>Lahey F77L-EM/32 3.01</td>
<td>1.6⁷⁸</td>
</tr>
<tr>
<td>25 MHz 386/387</td>
<td>Linux 1.2.13 (Unix)</td>
<td>f2c + gcc 2.7.0 -s -O2</td>
<td>2.5²⁷⁸</td>
</tr>
<tr>
<td>Award 33 MHz 386/387</td>
<td>DRDOS 6.0</td>
<td>Microsoft Powerstation 1.0</td>
<td>2.8¹⁸⁸</td>
</tr>
<tr>
<td>Compaq 20 MHz 386/Weitek 1167</td>
<td>OS386 1.0</td>
<td>Lahey F77L-EM/32 2.0</td>
<td>3.0¹⁸⁸</td>
</tr>
<tr>
<td>ACAD 25 MHz 486</td>
<td>OS386 2.1.04</td>
<td>Lahey F77L-EM/32 3.01</td>
<td>3.7¹⁸⁸</td>
</tr>
<tr>
<td>Micronics 33 MHz 386/Weitek 3167</td>
<td>OS386 1.9.16</td>
<td>Lahey F77L-EM/32 2.01</td>
<td>5.0¹⁸⁸</td>
</tr>
<tr>
<td>Dell 33 MHz 486E</td>
<td>OS386 2.1.04</td>
<td>Lahey F77L-EM/32 3.01</td>
<td>5.1⁷⁸</td>
</tr>
<tr>
<td>33 MHz 486DX</td>
<td>Windows 3.1 (foreground)</td>
<td>f2c + gcc 2.7.0 -s -O2</td>
<td>6.3¹⁸⁸</td>
</tr>
<tr>
<td>Acer 33 MHz 486DX</td>
<td>DRDOS 6.0</td>
<td>Microsoft Powerstation 1.0</td>
<td>6.3¹⁸⁸</td>
</tr>
<tr>
<td>AcerPower 33 MHz 486DX</td>
<td>Windows 3.1 (background)</td>
<td>f2c + Microsoft C/C++ 9.20 /O2</td>
<td>6.2²⁵⁸</td>
</tr>
<tr>
<td>AcerPower 33 MHz 486DX</td>
<td>MS-DOS 5.2</td>
<td>Lahey F77L-EM/32 5.20</td>
<td>6.5²⁵⁸</td>
</tr>
<tr>
<td>AcerPower 33 MHz 486DX</td>
<td>Linux 1.2.13 (Unix)</td>
<td>f2c + gcc 2.7.0 -s -O2</td>
<td>7.2²⁵⁸</td>
</tr>
<tr>
<td>66 MHz 486</td>
<td>Windows NT 3.51 WS</td>
<td>f2c + GNU-C 2.5.8</td>
<td>8.6²⁷⁸</td>
</tr>
<tr>
<td>66 MHz 486/DX2</td>
<td>Linux 1.1,54 (Unix)</td>
<td>f2c + GNU-C 2.5.8</td>
<td>11.2⁷⁸</td>
</tr>
<tr>
<td>Compaq Deskpro XL566 (60 MHz Pentium)</td>
<td>Windows NT 3.51 WS</td>
<td>f2c + Microsoft C/C++ 9.20 /O2</td>
<td>20³⁶⁸</td>
</tr>
<tr>
<td>Compaq Deskpro XE560 (60MHz Pentium)</td>
<td>MS-DOS 5.0/V</td>
<td>Lahey F77L-EM/32 5.20</td>
<td>21¹⁹⁸</td>
</tr>
<tr>
<td>Compaq Deskpro XE560 (60MHz Pentium)</td>
<td>MS-DOS 6.2/V</td>
<td>Lahey LF90-1.10b</td>
<td>23¹⁹⁸</td>
</tr>
<tr>
<td>Compaq Deskpro XE560 (60MHz Pentium)</td>
<td>MS-DOS 6.2/V</td>
<td>MS-Fortran Powerstation</td>
<td>25¹⁹⁸</td>
</tr>
<tr>
<td>90 MHz Pentium</td>
<td>Windows95/DOS</td>
<td>Lahey F77L-EM/32</td>
<td>26²⁰⁸</td>
</tr>
<tr>
<td>HP Vectra XU 5/90C (90 MHz Pentium)</td>
<td>Windows NT 3.5 WS</td>
<td>f2c + Microsoft C/C++ 9.20 /O2</td>
<td>26³⁶⁸</td>
</tr>
<tr>
<td>AcerPower 560p (60MHz Pentium)</td>
<td>MS-DOS V6.2</td>
<td>Lahey F77L-EM/32 5.20</td>
<td>28²⁵⁸</td>
</tr>
<tr>
<td>90 MHz Pentium</td>
<td>Windows95/DOS</td>
<td>Microsoft Powerstation 1.00 /G4/OX</td>
<td>33³⁵⁸</td>
</tr>
<tr>
<td>100 MHz Pentium</td>
<td>Linux 1.2.13 (Unix)</td>
<td>f2c + gcc 2.7.0 -s -O2</td>
<td>35²⁷⁸</td>
</tr>
<tr>
<td>Protech GoSet (150MHz Pentium - 256kB cache)</td>
<td>Windows 95</td>
<td>Lahey F790 2.00b -O1</td>
<td>40²⁴¹</td>
</tr>
<tr>
<td>Dell OmniPlex 590 (90MHz Pentium)</td>
<td>MS-DOS V6.2</td>
<td>Lahey F77L-EM/32 5.20</td>
<td>42²⁵⁸</td>
</tr>
<tr>
<td>Hitachi Flora 1010 (120MHz Pentium)</td>
<td>MS-DOS 6.2/V</td>
<td>MS-Fortran Powerstation</td>
<td>47¹⁹⁸</td>
</tr>
<tr>
<td>MACHINE</td>
<td>O/S</td>
<td>FORTRAN</td>
<td>RATIO</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------</td>
<td>---------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>VAX 11/780 FPA</td>
<td>VMS 3.7</td>
<td>FORT-11 3.7</td>
<td>1</td>
</tr>
<tr>
<td>DOS/Windows/NT machines</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASSUS TP4XEG w/PIPE 512k (150MHz Pentium)</td>
<td>Windows95</td>
<td>NPD 4.6.0 -u2 -u3 -norc -flic</td>
<td>49.38</td>
</tr>
<tr>
<td>ASSUS TP4XEG w/PIPE 512k (150MHz Pentium)</td>
<td>Linux 1.2.13</td>
<td>f2c + gcc -O2</td>
<td>51.38</td>
</tr>
<tr>
<td>GA-586ATE (133MHz Pentium)</td>
<td>Linux 1.2.12</td>
<td>f2c + gcc -m486 -static -O2</td>
<td>51.35</td>
</tr>
<tr>
<td>Dell Optiplex XMT5133 (133MHz Pentium)</td>
<td>MS-DOS 6.2/V</td>
<td>MS-Fortran Powerstation</td>
<td>53.18</td>
</tr>
<tr>
<td>HP Vectra XU 6/150 (150 MHz Pentium Pro)</td>
<td>Windows NT 3.51 WS</td>
<td>f2c + Microsoft C/C++ 9.20 /O2</td>
<td>70.36</td>
</tr>
<tr>
<td>Alpha Station 200 4/233 (233 MHz 21064A)</td>
<td>Windows NT 3.51 WS</td>
<td>f2c + Microsoft &amp; Digital AXP C/C++ 9.00 /O2</td>
<td>71.36</td>
</tr>
<tr>
<td>HP Vectra XU 6/150 (150MHz PentiumPro)</td>
<td>Windows NT 3.5</td>
<td>Microsoft Fortran Power Station 4.0 FL32 /Ox /W0 /G5</td>
<td>85.19</td>
</tr>
<tr>
<td>IBM DX2/66 + 4 MB cache</td>
<td>DOS 6.1/Windows 3.1</td>
<td>Lahey F77L-EM/32</td>
<td>98.23</td>
</tr>
<tr>
<td>200 MHz Pentium Pro (256k cache)</td>
<td>Linux 1.2.13 (Unix)</td>
<td>f2c + gcc 2.7,0 -s -O2</td>
<td>102.27</td>
</tr>
<tr>
<td>Unix Workstations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sun 3/60</td>
<td>SunOS 4.0</td>
<td>SunFortran 1.2 -f88881</td>
<td>0.905</td>
</tr>
<tr>
<td>HP-9000/370/Turbo SRX</td>
<td>HP-UX 6.5</td>
<td>HP Fortran V46.24(20)</td>
<td>1.93</td>
</tr>
<tr>
<td>SunSparcstation1</td>
<td>SunOS 4.0.3c</td>
<td>SunFortran 1.2 -O3</td>
<td>4.7</td>
</tr>
<tr>
<td>SunSparcstation1 +</td>
<td>SunOS 4.1.1</td>
<td>SunFortran 1.3 -fast -O4 -Bstatic</td>
<td>6.4</td>
</tr>
<tr>
<td>Decstation 5000/120</td>
<td>U-32 3.1</td>
<td>V1.0 -align</td>
<td>7.3</td>
</tr>
<tr>
<td>Oikodata Vistra 800 (40 MHz i860)</td>
<td>AT&amp;T System V</td>
<td>Portland Group PGF77-00</td>
<td>7.510</td>
</tr>
<tr>
<td>Decstation 5000/120</td>
<td>Ultrix 4.2a</td>
<td>V3.1</td>
<td>10.13</td>
</tr>
<tr>
<td>SG Personal IRIS 4D/25</td>
<td>IRIX 3.2.1</td>
<td>3.2 F77 1.31 -static</td>
<td>10</td>
</tr>
<tr>
<td>IBM R6000 Model 320</td>
<td>AIX 3.01</td>
<td>xlf 1.01 -Q -O -qornrdsnlg</td>
<td>11</td>
</tr>
<tr>
<td>SunSparcstation2</td>
<td>SunOS 4.1.1</td>
<td>SunFortran 1.3 -fast -O4 -Bstatic</td>
<td>11</td>
</tr>
<tr>
<td>SunSparcstation IPX</td>
<td>SunOS 4.1.1</td>
<td>SunFortran 1.3 -fast -O4 -Bstatic</td>
<td>12</td>
</tr>
<tr>
<td>IBM R6000 Model 530</td>
<td>AIX 3.01</td>
<td>xlf 1.01 -Q -O -qornrdsnlg</td>
<td>14</td>
</tr>
<tr>
<td>SG Indigo (33 MHz R3000)</td>
<td>IRIX 3.0</td>
<td>F77 3.4.1 -O3 -Olimit 2048 -static</td>
<td>17.34</td>
</tr>
<tr>
<td>SunSparcstation2 + Weitek µP</td>
<td>SunOS 4.1.3</td>
<td>SunFortran 1.4 -O4 -Bstatic</td>
<td>18.26</td>
</tr>
<tr>
<td>IBM R6000 Model 340</td>
<td>AIX 3.2</td>
<td>xlf 2.2 -Q -O -qorumdsnlg</td>
<td>18.17</td>
</tr>
<tr>
<td>HP-9000/710</td>
<td>HP-UX 9.01</td>
<td>HP Fortran 77 V9.00 -O3</td>
<td>19.12</td>
</tr>
<tr>
<td>SunSparcstation 10/41</td>
<td>SunOS 4.1.3,U1</td>
<td>SunFortran 1.4 -O4 -Bstatic</td>
<td>20.26</td>
</tr>
<tr>
<td>SunSparcstation 10/31</td>
<td>SunOS 4.1.3</td>
<td>SunFortran 1.3.1 -O4 -Bstatic</td>
<td>21.14</td>
</tr>
<tr>
<td>SG Personal IRIS 4D/35 (36 MHz R3000)</td>
<td>IRIX 5.3</td>
<td>F77 4.0,2 -mips1 -O3 -Olimit 2048 -static</td>
<td>21.28</td>
</tr>
<tr>
<td>MACHINE</td>
<td>O/S</td>
<td>FORTRAN</td>
<td>RATIO</td>
</tr>
<tr>
<td>---------</td>
<td>-----</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>VAX 11/780 FPA</td>
<td>VMS 3.7</td>
<td>FORT-11 3.7</td>
<td>≡ 1</td>
</tr>
<tr>
<td><strong>Unix Workstations</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HP-9000/720</td>
<td>HP-UX 8.0: pre-IC2</td>
<td>HP Fortran 77 V8.01 -K -O</td>
<td>24</td>
</tr>
<tr>
<td>HP-9000/715/50</td>
<td>HP-UX 9.03</td>
<td>HP-UX Fortran/9000 V9.0 -v +E1 -K +U77 +O3</td>
<td>25^n</td>
</tr>
<tr>
<td>Sun Sparstation 5 (? MHz µSparc II)</td>
<td>SunOS 5.4</td>
<td>SunFortran 3.0.1 -O4 -Bstatic</td>
<td>25^n</td>
</tr>
<tr>
<td>SGI Indigo (100 MHz R4000)</td>
<td>IRIX 4.0, 5F</td>
<td>F77 3.10 -O3 -mips2 -static -sopt</td>
<td>31</td>
</tr>
<tr>
<td>Sun Sparstation 10/51</td>
<td>SunOS 4.1.3 U1</td>
<td>SunFortran 1.4 -O4 -Bstatic</td>
<td>32^n</td>
</tr>
<tr>
<td>(Superscalar SPARC V8)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM RS/6000 Model 360</td>
<td>AIX 3.2.5</td>
<td>SunFortran 3.0.1 -O3 -Bstatic(?)</td>
<td>34^n</td>
</tr>
<tr>
<td>Sun Sparstation 10/52(?)</td>
<td>SunOS 5.3</td>
<td>SunFortran 1.4(? ) -O4(?) -Bstatic(?)</td>
<td>35^n</td>
</tr>
<tr>
<td>(Superscalar SPARC V8)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HP-9000/730</td>
<td>HP-UX 9.01</td>
<td>HP Fortran 77 V9.00 -O3</td>
<td>36^n</td>
</tr>
<tr>
<td>DEC AXP 3000/300L/300L (100 MHz)</td>
<td>OSF/1 V3.0 B</td>
<td>DEC Fortran V3.3 -opt4</td>
<td>37^n</td>
</tr>
<tr>
<td>HP-9000/712/80i (80 MHz PA7100LC)</td>
<td>HP-UX 9.07</td>
<td>HP Fortran 77 -K +O3</td>
<td>38^n</td>
</tr>
<tr>
<td>Sun Sparstation 20/712</td>
<td>Solaris 2.4</td>
<td>SunFortran 1.4 -O4</td>
<td>38^n</td>
</tr>
<tr>
<td>(2×75 MHz SuperSparc)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sun Sparstation 20/514</td>
<td>SunOS 5.3</td>
<td>SunFortran 3.0.1 -O4 -Bstatic</td>
<td>41</td>
</tr>
<tr>
<td>(4×75 MHz SuperSparc)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC AXP 3000/400</td>
<td>OSF/1 V1,3</td>
<td>DEC Fortran V3.3</td>
<td>42</td>
</tr>
<tr>
<td>IBM RS/6000 Model 370</td>
<td>AIX 3.2,3</td>
<td>xlf 2.03.0000.0012 -O3 -qsave</td>
<td>42^n</td>
</tr>
<tr>
<td>DEC AXP 3000/500</td>
<td>OSF/1 V1,2</td>
<td>DEC Fortran V3.3</td>
<td>43^n</td>
</tr>
<tr>
<td>IBM RS/6000 Model 3AT</td>
<td>AIX 3.25</td>
<td>F77 3.1 (opt4)</td>
<td>49^n</td>
</tr>
<tr>
<td>HP-9000/735</td>
<td>HP-UX 9.01</td>
<td>HP Fortran 77 V8.07 -K -O</td>
<td>53^n</td>
</tr>
<tr>
<td>DEC AXP 3000/600 (1 CPU)</td>
<td>OSF/1 V2.0</td>
<td>DEC Fortran V3.4</td>
<td>55^n</td>
</tr>
<tr>
<td>HP-9000/755</td>
<td>HP-UX 9.05</td>
<td>DEC Fortran V3.4</td>
<td>55^n</td>
</tr>
<tr>
<td>IBM R6000 Model 590</td>
<td>AIX 3.25</td>
<td>DEC Fortran -fnst</td>
<td>58^n</td>
</tr>
<tr>
<td>DEC AXP 175 MHz</td>
<td>OSF/1 V3.2 (Rev. 214)</td>
<td>DEC Fortran -fnst</td>
<td>58^n</td>
</tr>
<tr>
<td>SGI Indigo (150 MHz R4400)</td>
<td>IRIX 5.2</td>
<td>F77 4.0.1 -O3 -mips2 -static -sopt -non_shared</td>
<td>61</td>
</tr>
<tr>
<td>SGI Indy/SC (150 MHz R4400)</td>
<td>IRIX 5.2</td>
<td>F77 4.0.1 -O3 -mips2 -static -sopt -non_shared</td>
<td>61</td>
</tr>
<tr>
<td>HP-9000/C110 (120 MHz PA7200)</td>
<td>HP-UX 9.07</td>
<td>f2c + HP-UX C compiler +O3</td>
<td>62^n</td>
</tr>
<tr>
<td>IBM RS/6000 Model 390</td>
<td>AIX 3.</td>
<td>xlf 3.02.0000.0000 -O3 -qarch=pwr2 -qsave</td>
<td>62^n</td>
</tr>
<tr>
<td>SGI TFP (75 MHz R8000)</td>
<td>IRIX64 6,1</td>
<td>F77 6.1 -O3 -static</td>
<td>66^n</td>
</tr>
<tr>
<td>MACHINE</td>
<td>Normalization</td>
<td>O/S</td>
<td>FORTRAN</td>
</tr>
<tr>
<td>----------</td>
<td>---------------</td>
<td>-----</td>
<td>---------</td>
</tr>
<tr>
<td>VAX 11/780 FPA</td>
<td></td>
<td>VMS 3.7</td>
<td>FORT11 3.7</td>
</tr>
<tr>
<td><strong>Unix Workstations</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC AXP 3000/800 (200 MHz)</td>
<td></td>
<td>OSF/1 V2.0 (Rev. 240)</td>
<td>DEC Fortran V3.3-342 -O4 -static</td>
</tr>
<tr>
<td>HP-9000/735-125</td>
<td></td>
<td>HP-UX 9.05</td>
<td>HP Fortran 77 -K +O3</td>
</tr>
<tr>
<td>HP-9000/C110 (120 MHz PA7200)</td>
<td></td>
<td>HP-UX 9.07</td>
<td>HP Fortran 77 -K +O3</td>
</tr>
<tr>
<td>SGI TFP (90 MHz R8000)</td>
<td></td>
<td>IRIX 6.1</td>
<td>F77 6.1 -O3 -static</td>
</tr>
<tr>
<td>SGI Indigo^2 (200 MHz MIPS R4400)</td>
<td></td>
<td>IRIX 6.2</td>
<td>F90 6.2 -n32 -mips3 -O3 -static</td>
</tr>
<tr>
<td>SGI Indigo^2 (200 MHz MIPS R4400)</td>
<td></td>
<td>IRIX 6.2</td>
<td>F77 6.2 -mips2 -O3 -static</td>
</tr>
<tr>
<td>DEC AXP 233 MHz</td>
<td></td>
<td>OSF/1 V5.6</td>
<td>DEC Fortran -fast</td>
</tr>
<tr>
<td>Sun Ultra 1 (167 MHz SPARC)</td>
<td></td>
<td>SunOS 5.5</td>
<td>SunFortran 4.0 -O4 -Bstatic</td>
</tr>
<tr>
<td>DEC AXP 250-4/266 (266 MHz 21064)</td>
<td></td>
<td>OSF v 3.2c</td>
<td>DEC Fortran 3.8 -O4 -static</td>
</tr>
<tr>
<td>DEC AXP 600-5/266 (266 MHz 21164)</td>
<td></td>
<td>OSF v 3.2c</td>
<td>DEC Fortran 3.8 -O4 -static</td>
</tr>
<tr>
<td>Microway Screamer (DEC 300 MHz 21164)</td>
<td></td>
<td>DEC UNIX V 3.2C Rev 148</td>
<td>DEC Fortran V 3.8</td>
</tr>
<tr>
<td>SGI Indigo Impact (195 MHz MIPS R10000)</td>
<td></td>
<td>IRIX 6.2</td>
<td>F77 6.2 -O3 -mips4 -static -sopt</td>
</tr>
<tr>
<td><strong>Non-Unix Workstations</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAXstation 31/30</td>
<td></td>
<td>VMS 5.1</td>
<td>FORT11 5.3</td>
</tr>
<tr>
<td>VAXstation 31/38</td>
<td></td>
<td>VMS 5.3</td>
<td>FORT11 5.3</td>
</tr>
<tr>
<td>VAXstation 31/76</td>
<td></td>
<td>VMS 5.4</td>
<td>FORTRAN V5.8-155</td>
</tr>
<tr>
<td>DEC AXP 3000/400</td>
<td></td>
<td>OpenVMS Alpha 1.5</td>
<td>DEC Fortran 6.1</td>
</tr>
<tr>
<td>DEC AXP 3000/500</td>
<td></td>
<td>OpenVMS Alpha 1.5</td>
<td>DEC Fortran 6.1</td>
</tr>
<tr>
<td><strong>Unix Minis/Servers</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sun 4/330</td>
<td></td>
<td>SunOS 4.0.3</td>
<td>SunFortran 1.2</td>
</tr>
<tr>
<td>SGI 280S (1 CPU - 25 MHz)</td>
<td></td>
<td>IRIX 3.2,1</td>
<td>3.2 F77 1.31 -static</td>
</tr>
<tr>
<td>SGI 280S (1 CPU - 25 MHz)</td>
<td></td>
<td>IRIX 4.0,1</td>
<td>3.41 F77 -O3 -static</td>
</tr>
<tr>
<td>SGI 280S (1 CPU - 33 MHz)</td>
<td></td>
<td>IRIX 4.0,1</td>
<td>3.41 F77 -O3 -static</td>
</tr>
<tr>
<td>DEC AXP 4000/710 (1 CPU)</td>
<td></td>
<td>OSF/1 V1.3</td>
<td>DEC Fortran V3.3</td>
</tr>
<tr>
<td>HP-9000/J200 (1×100 MHz PA7200)</td>
<td></td>
<td>HP-UX 9.05</td>
<td>HP Fortran 77 -K +O3</td>
</tr>
<tr>
<td>SGI Challenge (150 MHz R8000)</td>
<td></td>
<td>IRIX 5.3</td>
<td>F77 4.0,2 -mips2 -O3 -Olimit 2048 -static</td>
</tr>
<tr>
<td>SGI Power Challenge (90 MHz R4400)</td>
<td></td>
<td>IRIX 6.0,1</td>
<td>MipsPro F77 6.0.1 -mips4 -O3 -static</td>
</tr>
<tr>
<td><strong>Non-Unix Minis/Servers</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>µVax II</td>
<td></td>
<td>VMS 5.5</td>
<td>FORT11 5.8</td>
</tr>
<tr>
<td>VAX 11/780 FPA</td>
<td></td>
<td>VMS 4.5</td>
<td>FORT11 5.3</td>
</tr>
<tr>
<td>MACHINE</td>
<td>O/S</td>
<td>FORTRAN</td>
<td>RATIO</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>--------</td>
</tr>
<tr>
<td>VAX 11/780 FPA</td>
<td>VMS 3.7</td>
<td>FORT-11 3.7</td>
<td>1</td>
</tr>
<tr>
<td><strong>Non-Unix Minis/Servers</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>μVax 3300</td>
<td>VMS 5.3</td>
<td>FORT-11 5.3</td>
<td>1.6⁶²</td>
</tr>
<tr>
<td>μVax 3600</td>
<td>VMS 5.4</td>
<td>FORTRAN V5.8-155</td>
<td>2.4⁵⁰</td>
</tr>
<tr>
<td>μVax 3600</td>
<td>VMS 4.7</td>
<td>FORT-11 4.7a</td>
<td>2.4⁶⁰</td>
</tr>
<tr>
<td>μVax 3800</td>
<td>VMS 5.1</td>
<td>FORT-11 5.3</td>
<td>2.8⁶²</td>
</tr>
<tr>
<td>μVax 3100</td>
<td>VMS 5.5</td>
<td>FORT-11 5.4</td>
<td>3.0⁴⁴</td>
</tr>
<tr>
<td>μVax 3800</td>
<td>VMS 5.4</td>
<td>FORTRAN V5.8-155</td>
<td>3.2⁶⁰</td>
</tr>
<tr>
<td>VAX 4000/60</td>
<td>VMS 5.5</td>
<td>FORTRAN 5.8</td>
<td>3.2⁴⁰</td>
</tr>
<tr>
<td>VAX 4000/600</td>
<td>VMS 5.5</td>
<td>FORTRAN 5.8</td>
<td>3.2⁴⁰</td>
</tr>
<tr>
<td>HP NetServer 5/133 LS4 (1 of 4 133 MHz Pentium CPUs)</td>
<td>Windows NT 3.51 Server</td>
<td>f2c + Microsoft C/C++ 9.20 /O2</td>
<td>54⁶²⁰</td>
</tr>
<tr>
<td><strong>Mainframes</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAX 8200</td>
<td>VMS 5.3</td>
<td>FORT-11 5.3</td>
<td>0.7⁷⁶</td>
</tr>
<tr>
<td>Vax 6330 (1 CPU)</td>
<td>VMS 5.2</td>
<td>FORT-11 5.2-33</td>
<td>3.4²</td>
</tr>
<tr>
<td>IBM 3090/3090E</td>
<td>VM/HPO 4.2 CMS</td>
<td>VSFORT 2.0</td>
<td>24/25</td>
</tr>
<tr>
<td><strong>Supercomputers</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cray T3D (1 CPU, 6.6 ns)</td>
<td>UNICOS-MAX 1.2.0.3</td>
<td>Cray CFT77-M Version 6.2.1.3</td>
<td>17³²²</td>
</tr>
<tr>
<td>Cray J90 (1 CPU, 10 ns)</td>
<td>UNICOS 8.0.4</td>
<td>Cray CFT77 Version 6.0.4</td>
<td>19³²</td>
</tr>
<tr>
<td>Cray YMP (1 CPU, 6 ns)</td>
<td>UNICOS 8.0.3</td>
<td>Cray CFT77 Version 6.0.4</td>
<td>36³²</td>
</tr>
<tr>
<td>Cray C90 (1 CPU, 4.167 ns)</td>
<td>UNICOS 8.0.4</td>
<td>Cray CFT77 Version 6.0.4</td>
<td>52³²²</td>
</tr>
<tr>
<td>Fujitsu VP2200/10 (scalar mode)</td>
<td>UXP/M V10L10</td>
<td>Fujitsu FORTRAN77 EX V12</td>
<td>58¹⁵</td>
</tr>
<tr>
<td><strong>Exotica</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meiko (1×T800 20 MHz)</td>
<td>MMVCS 1.4</td>
<td>Meiko F77 1,100</td>
<td>2.⁴²</td>
</tr>
<tr>
<td>nCUBE (1×Custom VLSI 64-bit)</td>
<td>SunOS 4.1.1 (EARTH CUBE)#10</td>
<td>nCUBE2 Fortran 3.2 (no optimization)</td>
<td>3.⁸²⁰</td>
</tr>
<tr>
<td>BBN TC2000 (1×MC88000 20Mhz)</td>
<td>UNIX 4.3BSD + pSOS++m</td>
<td>Fortran-88000 3.45 -OLM -Jnone -gg</td>
<td>8.⁶⁸⁰</td>
</tr>
<tr>
<td>SkyBolt (860 40 MHz)</td>
<td>Host</td>
<td>SKYvec F77</td>
<td>10⁶⁵</td>
</tr>
<tr>
<td>KSR1 (1×KSR1 processor 20MHz)</td>
<td>KSR OS 1.0.6</td>
<td>KSR Fortran f77 -O2 -inline -from * .il</td>
<td>11³¹</td>
</tr>
<tr>
<td>SPARC Center 1000 (1 of 8 CPU's only)</td>
<td>SunOS 5.4</td>
<td>f77 -O4</td>
<td>50⁵⁰¹</td>
</tr>
</tbody>
</table>

This benchmark is maintained by:
Alex F Bidajew
Institute for National Measurement Standards
National Research Council of Canada
ACKNOWLEDGEMENTS

Many people have contributed to this benchmark study. The original benchmark code was created by Dave Rogers (NRC). Individual contributions are denoted by numerical superscripts beside the results in the Table. They are:

1. Sue Walker (IAPM, Seattle, USA)
2. Simon Duane (NPL, London, GB)
3. Juhani Heinila (TRC-MEL, Tampere, Finland)
4. Jaap Davelaar (AZL, Leiden, Holland)
5. Mark Looper (SRL-CIT, Pasadena, USA)
6. Pedro Andreo (Karolinska Institute, Stockholm, Sweden)
7. Donald Johnson (Moncrief Radiation Center, Fort Worth, Texas, USA)
8. William Celmaster (BBN Advanced Computer Systems, Cambridge, Massachusetts, USA)
9. D. Douglas Wilkinson (Sky Computers, Chelmsford, Massachusetts, USA)
10. David Gladstone (Harvard Medical School, Boston, Massachusetts, USA (Note that PGF77 would not compile XYZDOS at higher optimization.)
11. Howard Page (Kendall Square Research, Waltham, Massachusetts, USA)
12. Pertti Henttu (Oulu University Central Hospital, Oulu, Finland)
13. Peter Hoban (Royal Adelaide Hospital, Adelaide, Australia)
14. David Murray (IGE Medical Systems, Herts, UK)
15. Paul Miskedy (Biomedicine & Health Program, Ansto, Menai, Australia)
16. Michel Proulx (National Research Council of Canada)
17. Joe Heamge (UCSF Physics Research Lab, USA)
18. Stephen Pistorius (Manitoba Cancer Treatment and Research Foundation, Canada)
19. Yoshihito Namito (KEK High Energy Physics Laboratory, Japan)
20. Jeff Chen (London Regional Cancer Centre, London, Canada)
21. Alexandr Mahasek (Czech Technical University in Prague, Czech Republic)
22. Coghe Marc (Academic Hospital Free University Brussels, Belgium)
23 Barone Rosso (Aiko Enterprises, Elinorville, USA)
24 Richard Knight (Royal Marsden Hospital, Surrey, UK)
25 Gary Goozee (St George Hospital, Sydney, Australia)
26 Kees-Jan Leliyeld (KSEPL/FOM, Rijswijk, The Netherlands)
27 Petri Sipila (Finnish Centre for Radiation and Nuclear Safety, Helsinki, Finland)
28 Doug Ritchie (National Research Council of Canada)
29 Arnold Aronson (Brookhaven National Laboratory, USA)
30 Lionel Boachet (Department of Nuclear Engineering, University of Florida, USA)
31 Jan Senftjens (Department of Biomedical Physics, University of Gent, Belgium)
32 Gabriel Thabet (Cray Research Inc, France)
33 Jann Boucek (SCGH, Perth, Australia)
34 Erik Korevaar (Dr Daniel den Hoed Cancer Centre, Rotterdam, the Netherlands)
35 Matthias Fippel (U Leipzig, Germany)
36 Joakim Laitinen (Varian-Dosetek, Finland)
37 Pierre De Lit (Brussels University, Belgium)
38 Iwan Kawrakow (INMS/NRC, Canada)
39 Nina Nitroy (Microway Inc, USA)
40 Jerome Goutichon (GE Medical Systems, France)
41 Martin Caon (Flinders University, Australia)
42 Anita Rao (Siemens OCS, Concorde, California)
2.3 Single or double precision?

For most modern architectures, it does not cost very much extra to run applications in double rather than single precision with the possible exception of the VAX. This feature is true for their entire product line with double precision overheads ranging from 20%–100%! Some architectures do not even support single precision floating-point numbers (e.g. FPS serial processors) and other architectures, while supporting single-precision arithmetic, do so with some associated overhead (e.g. IBM RS6000's)!

The author has found significant differences in comparing calculated results of stopping powers (significant to the fifth digit) from single precision simulations. Differences in energy deposition accumulation (good to about 3 digits) begin to take effect at about $10^6$–$10^7$ histories depending on whether the energy deposition increment is constant or random. An example of accumulation error is shown in fig. 1. For constant increment, $s = s + \sum_{i=1}^{N} (1/N)$, noticeable accumulation error is seen starting from about $10^6$ iterations. The shape of this curve is the result of constant accumulated round-off error, can be positive or negative, but eventually underestimates due to truncation error. (In single precision arithmetic, $1 + 10^{-8} \approx 1$). The shape of this curve is architecture-dependent. (This simulation was performed on a Sparc/IPX machine.) For random increment, $s = s + \sum_{i=1}^{N} \xi_i (1/N)$ where $0 < \xi_i < 2$, noticeable accumulation error is seen starting from about $10^6$ iterations. The shape of this curve is the result of random accumulation error and is common to all single-precision architectures. A double precision accumulation is shown as well. For double precision, $1 + 10^{-8} = 1.00000001$ and no accumulation error is evident.

Ray tracing of particles is done better in double precision. Unless the geometry of curved surfaces is coded very carefully, round-off and truncation of floating-point numbers can cause difficulty. The rule of thumb is: If you can afford it, run in double precision.

2.4 What language should I program in?

EGS4 is written in MORTAN3, a FORTRAN pre-compiler that can convert the EGS4 MORTAN3 source code into FORTRAN ANSI standard 77 or 66 code. There is no vacillation regarding MORTAN3. There are those who HATE it and those who LOVE it. If you prefer you can write all your code in FORTRAN although it will take you longer and it will be difficult to make your code readable. To compensate for all the mistakes you make you will have direct access to FORTRAN debugging tools. Experienced MORTAN3 programmers write more readable, more compact code than is possible in FORTRAN. On the downside, it is more difficult to debug and you use debugging tools indirectly since references to FORTRAN statements are only indirectly related to the MORTAN source.

To give you an example: The NRCC statistical analysis code takes about 100 lines of MORTAN3 source (including comments) which translates into about 600 lines of FORTRAN (with no comments). This code was conceived, written and debugged in one working day. It is IMPOSSIBLE to conceive, write and debug 600 lines of FORTRAN code in one day.

It is also possible to write usercodes in any language (e.g. C, C++) and communicate with EGS using standard interface routines. The future may have EGS being distributed in standard C with a MORTAN-like pre-compiler to include all the programming power inherent in MORTAN.

2.5 Should I use non-standard FORTRAN?

Avoid non-standard FORTRAN coding like the plague. Although all compilers have
Figure 1: An example of constant and random accumulation errors in single-precision arithmetic.
FORTRAN-extensions that make coding faster and more readable, you will duplicate your efforts when you switch to a different machine. There are differences you can not avoid. They are:

- System timing calls like CPU time used, total elapsed time, time and date routines. Unfortunately, these are not standardised like mathematics subroutines. For example, VAX/VMS machines can obtain the date via the call
  
  \texttt{CALL DATE(DATEN)}
  
  where \texttt{DATEN} is a \texttt{CHARACTER*9} variable while IBM/CMS machines obtain the date via
  
  \texttt{CALL DATE(DAQFWK, DATEN)}
  
  where \texttt{DAQFWK} is a \texttt{CHARACTER*8} variable (day of the week) and \texttt{DATEN} is a \texttt{CHARACTER*8} variable. Note that in this case, unless you knew to make the change, the compiler would not complain (most FORTRAN's do not check subroutine interfaces) and you would get unexpected results.

- I/O operations are not standardised. For example, to open a file with a specified name in VAX/VMS one issues the call
  
  \texttt{OPEN(...NAME='filename'...)}
  
  and on IBM/CMS
  
  \texttt{OPEN(...FILE='filename'...)}.
  
  There are a number of non-standard qualifiers for both \texttt{OPEN}, \texttt{CLOSE}, \texttt{READ} and \texttt{WRITE} statements.

- Be aware of the differences in machine precision. For example, a PRESTA single precision convergence routine can work to 1 part in $10^7$ on the VAX but only 5 parts in $10^7$ on an IBM owing to differences in floating-point binary representations.

- Pseudo random number generation usually depends upon machine architecture. This topic is dealt with in detail in the next section.

If you use different machines it is easy to write macros that can covert your code to run on any machine and design user codes with this general application in mind.

The UNIX FORTRAN world has made generous concessions to VAX/VMS FORTRAN. Most UNIX-based FORTRAN compilers allow many of the VAX/VMS FORTRAN extensions (which are seductive in their simplicity and power). However, other conversions are not available (yet). Yet, the portability of standard ANSI code to other architectures should be motivation enough for adoption of standard coding practice.

3 Random number generators, state-of-the-art \textit{ca.} 1988

The pseudo random number generator is the “soul” of a Monte Carlo calculation. It is what generates the pseudo-random nature of Monte Carlo simulations thereby imitating the true stochastic nature of particle interactions. Consequently, much mathematical study has been devoted to RNG’s [3, 4]. The paper by Marsaglia [3] is the seminal work on the topic, allowing one to determine when one had a “good” RNG, while Knuth’s book [4] reviews the field and supplies a set of good RNG’s.

The operative phrase to be used is, “Use Extreme Caution.”

- \textbf{DON’T FIDDLE} with RNG’s unless you thoroughly understand the underlying mathematics and have the ability to test your new RNG completely.

- \textbf{DON’T TRUST} RNG’s that come bundled with standard mathematical packages. Scientists have wasted \textbf{YEARS} of their life trying to understand bugs in their Monte Carlo code that were really due to bad RNG’s.
• **DO USE** the RNG’s that are distributed with the EGS4 code. They have been thoroughly tested. Employing “in-line” RNG’s à la EGS4 will also save you about 20% CPU time. (Some modern compilers will automatically make subroutines “in-line” at high levels of compiler optimisation.) The overhead costs of function and subroutine calls is surprisingly large (about 20–100 machine cycles).

The gathering of random numbers into planes is a well known artefact of the type of RNG that EGS4 employs by default. Marsaglia’s paper [3] describes how random numbers fall into \((n - 1)\)-hyperplanes when seeding \(n\)-dimensional hypercubes (for \(n > 2\)). An example of a bad RNG with this characteristic is shown in fig. 2. This RNG was actually distributed with standard mathematical packages provided with mainframe and mini-computers. However, this artefact can be seen only under special circumstances. If the view angle is rotated by only 10 degrees as in fig. 3, the artefact is not apparent. Good RNG’s maximise the number of planes giving the illusion of randomness. The one distributed with EGS4 code gives over 1000 planes in a 3D cube.

3.1 Linear congruential RNG’s

The standard method of random number generation is the linear congruential random number generator (LCRNG). It has the form:

\[ X_n = (aX_{n-1} + b) \mod 2^k, \]

producing a series of pseudo random integers, \(X_1, X_2, X_3\ldots\) starting from a “seed” \(X_0\). The factor \(a\) is the constant multiplier, \(b\) is also a constant and \(k\) is usually the integer word size of the computer. If \(b\) is an odd number, the sequence length of this RNG is \(2^k\). Standard EGS4 sets \(b = 0\). This version is called a multiplicative congruential (MC) RNG with a sequence length of \(2^{k-2}\). The constant multiplier \(a\) is a *magic* number. **DO NOT** change it unless you know what you are doing. There are guidelines for choosing potentially good candidates, but every potential multiplier has to be tested “experimentally”. For \(k = 32\), EGS4 employs \(a = 663608941\). Knuth [4] claims that \(a = 69069\) (much easier to remember) is best. A particularly bad one is \(a = 65539\) which was employed to produce the data in figs. 2 and 3.

LCRNG’s “use up” every integer represented by the computer in a well-defined order and then start over. MCRNG’s “use up” half of the odd integers in sequence. Therefore, it is unwise to use up more than, say \(1/10^\text{th}\), of the sequence. For example, the last half of the sequence will be anticorrelated with the first half. If one requires more than \(10^8\) random numbers for a simulation (not an uncommon requirement) then one is well advised to employ long sequence random number generators described in the next sections.

Actually, a given Monte Carlo history may consume thousands of random numbers. It is important, if the RNG cycles, that no history starts with the same random number as a previous one. Rejection techniques, which “discard” random numbers, could “synchronise” a recycled RNG. It is difficult and expensive to test for this and much safer to employ a long sequence RNG.

3.2 Generic 32 and 64-bit RNG’s

The following MORTFRAN3 code will produce a \(2^{30}\) (about \(10^9\)) sequence of random numbers on any 32-bit computer with two’s-complement integer circuitry (i.e. hard-wired
Marsaglia planes – View 1

Figure 2: An example of the Marsaglia artefact produced with a catastrophically bad RNG. In this case a 3D unit cube was seeded and the data gathers into 15 planes.

Marsaglia planes – View 2

Figure 3: This is the same data as the previous figure only with a 10 degree difference in view angle.
integer arithmetic operations) allowing integer overflows to occur. (You may have to suppress certain classes of arithmetic exceptions. For example, on VAX/VMS one employs the 
\texttt{/CHECK=NOOVERFLOW} in the FORTRAN compilation command.)

"Initialisation:"
Ixx=9876543421; "Default RNG seed, declare IXX in common block RANDOM"

"Iteration:"
Ixx=Ixx*663608941; "Common RANDOM must be declared where used"
Rng=0.5*Ixx*0.23283064E-09; "0.23283064E-09 IS 2**(-32)"

This RNG produces the following output on a VAX:

\begin{verbatim}
Ixx,old =  987654321 Ixx,new = -1879502499 rng = 6.2394232E-02 
Ixx,old = -1879502499 Ixx,new =  89393817 rng = 0.5208136 
Ixx,old =  89393817 Ixx,new = 1262367013 rng = 0.7939177 
Ixx,old = 1262367013 Ixx,new = -1401290047 rng = 0.1737368 
Ixx,old = -1401290047 Ixx,new =  698583597 rng = 0.6626517 
Ixx,old =  698583597 Ixx,new = -2109875415 rng = 8.7563396E-03 
Ixx,old = -2109875415 Ixx,new =  217636469 rng = 0.5506724 
Ixx,old =  217636469 Ixx,new = -1336857135 rng = 0.1887387 
Ixx,old = -1336857135 Ixx,new = -1956069379 rng = 4.4567108E-02 
Ixx,old = -1956069379 Ixx,new =  2070947001 rng = 0.9821799 
Ixx,old =  2070947001 Ixx,new =  930943173 rng = 0.7167521 
Ixx,old =  930943173 Ixx,new =  1414265313 rng = 0.8292843 
Ixx,old = 1414265313 Ixx,new = -1359510835 rng = 0.1834642 
Ixx,old = -1359510835 Ixx,new =  1756985161 rng = 0.9090800 
Ixx,old = 1756985161 Ixx,new =  1050548245 rng = 0.7445998 
Ixx,old = 1050548245 Ixx,new =  952334065 rng = 0.7217326 
Ixx,old =  952334065 Ixx,new = -434396515 rng = 0.3988592 
Ixx,old = -434396515 Ixx,new =  1816224473 rng = 0.9228727 
Ixx,old = 1816224473 Ixx,new =  1276396645 rng = 0.7971843 
Ixx,old = 1276396645 Ixx,new =  110212353 rng = 0.5256608 
Ixx,old = 110212353 Ixx,new = -7475347 rng = 0.4982595 
Ixx,old = -7475347 Ixx,new =  1595438953 rng = 0.8714671 
Ixx,old = 1595438953 Ixx,new =  2026694069 rng = 0.9718765 
Ixx,old = 2026694069 Ixx,new =  908182545 rng = 0.7114527 
Ixx,old = 908182545 Ixx,new = -389793475 rng = 0.4092441 
Ixx,old =-389793475 Ixx,new =  382223609 rng = 0.5889934 
Ixx,old =  382223609 Ixx,new =  1262111749 rng = 0.7938583 
Ixx,old = 1262111749 Ixx,new =  931116065 rng = 0.7167923 
Ixx,old = 931116065 Ixx,new = -1665052659 rng = 0.1123247 
Ixx,old =-1665052659 Ixx,new =  1944279945 rng = 0.9526880 
Ixx,old =  1944279945 Ixx,new = -895236267 rng = 0.2915616 
Ixx,old = -895236267 Ixx,new =  191403313 rng = 0.5445645 
Ixx,old =  191403313 Ixx,new =  803537885 rng = 0.6870883 
Ixx,old =  803537885 Ixx,new = -912962791 rng = 0.2874343 
Ixx,old =-912962791 Ixx,new =  2004199333 rng = 0.9666390
\end{verbatim}
Ixx, Old = 2004199333  Ixx, New = 1566209857  RNG = 0.8646617
Ixx, Old = 1566209857  Ixx, New = -927193939  RNG = 0.2841209
Ixx, Old = -927193939  Ixx, New = 775070633  RNG = 0.6804602
Ixx, Old = 775070633  Ixx, New = 2022784245  RNG = 0.9709662
Ixx, Old = 2022784245  Ixx, New = 782527057  RNG = 0.6821963
Ixx, Old = 782527057  Ixx, New = 1439728253  RNG = 0.8352129
Ixx, Old = 1439728253  Ixx, New = 1154020665  RNG = 0.7686914
Ixx, Old = 1154020665  Ixx, New = 1302124357  RNG = 0.8031745
Ixx, Old = 1302124357  Ixx, New = 578190945  RNG = 0.6346206
Ixx, Old = 578190945  Ixx, New = -797381299  RNG = 0.3143452
Ixx, Old = -797381299  Ixx, New = 263114697  RNG = 0.5612612
Ixx, Old = 263114697  Ixx, New = 1448330901  RNG = 0.8372158
Ixx, Old = 1448330901  Ixx, New = 2062664561  RNG = 0.9802516
Ixx, Old = 2062664561  Ixx, New = -592890595  RNG = 0.3619569
Ixx, Old = -592890595  Ixx, New = 1544799065  RNG = 0.8596766

The employment of generic RNG’s allows one to synchronise histories on different machines (insofar as differences in floating point precision allow this to happen), which may be useful for debugging purposes.

Some processors mock-up integer arithmetic using floating-point processors. These machines require special code. An example is given later for the discussion of the FPS-264.

If 64-bit two’s-complement integer arithmetic were available, the following formula would serve as a generic 64-bit RNG:

\[
\begin{align*}
\text{Ixx}_64 &= 6364136223846793005 \times \text{Ixx}_64; \\
\text{RNG}_64 &= 0.5 + \text{Ixx}_64 \times 5.421010862E-20; \quad \text{"}5.421010862E-20 \text{ IS } 2^{63}\" \\
\end{align*}
\]

This RNG produces a sequence length of \(2^{62}\) (about \(4.6 \times 10^{18}\)). This sequence length should be long enough for any practical problem. This RNG requires machine dependent coding since INTEGER*8 arithmetic is not standard FORTRAN. An example for the VAX is given following.

### 3.3 VAX RNG’s

VAX employs the generic 32-bit RNG. The 64-bit version described above takes the form [5]:

"Declarations:"

```
COMMON/RANDM1/LNGX1;
COMMON/RANDM2/LNGX2;
COMMON/RANDM3/XAMSK;
COMMON/RANDM4/FFOUR,FTWO;
REAL*8 LNGX1,LNGX2,ZERO8;
INTEGER*4 LNGX2(2),FFOUR(3),LNGX2(2),WKREA1,WKREA2,WKREA3,XAMSK(3),Ixx,Jxx;
INTEGER*2 LNGX4(4),FTWO(4),LNGX4(4),XAMSK1,XAMSK2,XAMSK3;
EQUIVALENCE (LNGX1,LNGX2),(LNGX2,LNGX4);
EQUIVALENCE (LNGX2,LNGX2),(LNGX2,LNGX4);
EQUIVALENCE (LNGX4(1),WKREA1);
EQUIVALENCE (LNGX4(1),WKREA2);
```
EQUIVALENCE (LNGXB4(3), WKREA3);
EQUIVALENCE (XAMSK(1), XAMSK1);
EQUIVALENCE (XAMSK(2), XAMSK2);
EQUIVALENCE (XAMSK(3), XAMSK3);
EQUIVALENCE (XAMSK(1), ZERO8);
EQUIVALENCE (IXX,LNGXA2(1));
EQUIVALENCE (JXX,LNGXA2(2));

"Initialise the constant multiplier:"
DATA FFOUR/Z00007F2D, Z00004C95, Z0000F42D/;
DATA FTWO/Z7F2D, Z4C95, ZF42D, Z5851/;

"Initialise the seed:"
DATA IXX, JXX/987654321, 987654321/;

"Select a new random number:"
ZERO8=0.0D0;
XAMSK(3)=0;
WKREA3=0;
XAMSK1=LNGXA4(1);
XAMSK2=LNGXA4(2);
XAMSK3=LNGXA4(3);
WKREA1=FFOUR(1) * XAMSK(1);
WKREA2=WKREA2+FFOUR(1) * XAMSK(2);
WKREA3=WKREA3+FFOUR(1) * XAMSK(3);
LNGXB4(4)=LNGXB4(4)+FTWO(1) * LNGXA4(4);
WKREA2=WKREA2+FFOUR(2) * XAMSK(1);
WKREA3=WKREA3+FFOUR(2) * XAMSK(2);
LNGXB4(4)=LNGXB4(4)+FTWO(2) * LNGXA4(3);
WKREA3=WKREA3+FFOUR(3) * XAMSK(1);
LNGXB4(4)=LNGXB4(4)+FTWO(3) * LNGXA4(2);
LNGXB4(4)=LNGXB4(4)+FTWO(4) * LNGXA4(1);
LNGXA1=LNGXB1;
RNG=0.5+0.23283064E-09*LNGXB2(2);

One can see that much calculation is required to simulate 64-bit integer arithmetic and handle overflow properly. Use of this RNG slows down a typical simulation by approximately 20%. As a rule of thumb, we used to employ this RNG for any simulation using more than about 10⁶ histories. However, the lagged-Fibonacci RNG (described below) has supplanted the 64-bit RNG for all long-sequence calculations.

3.4 IBM RNG's

The 32-bit IBM RNG supplied with EGS4 makes elegant use of IBM's machine instructions for doing unnormalised floating point arithmetic. In effect, the following code prenormalises a floating point double precision number and scrambles the low-order 32-bits via integer random number generation.

"Declarations:"
INTEGER IXX; "IXX should reside in a common block"
INTEGER JX(2); REAL*8 DRN; "These are defined in the local subroutine"
EQUIVALENCE(JX(1),DRN);

"Initialisation:"
DATA JX(1)/Z46000000/;
IXX=987654321;

"Iteration:"
IXX=IXX*663608941; JX(2)=IXX; RNG=DRN+0.0D0;

This method employs four memory fetches, one integer multiplication, one double precision addition (to normalise the floating point number) and three stores. The IBM could as easily use the 32-bit generic method that requires four memory fetches, one integer multiplication, one integer-to-floating-point conversion, one floating point multiplication, one floating point addition and two stores. Clearly, the IBM special coding saves one numerical data type conversion and a floating point multiplication and is clearly more economical.

3.5 FPS-264 RNG

The FPS-264 is an example of a machine that does not have integer arithmetic. It is simply a double precision number cruncher. It does not even support single precision floating point operations. The FPS-264 FORTRAN handles integer operations like floating point operations within the 53-bit mantissa and the high order bits “spilled” like normal integer operations. (This is not documented in the FPS manuals!) The following RNG simulates 48-bit arithmetic employing the 48-bit multiplier times 32. This fills the 53-bit mantissa with the low order bits of the integer multiplication. The 53-bit integer is then normalised to fit in the range 0 <RNG< 1 by multiplication by $2^{-53}$. The random integer is then shifted so that the low order 48 bits of the integer multiplication are properly aligned.

"Declarations:"
INTEGER IXX; "Store in common. FPS treats as a REAL*8"

"Initialisation:"
IXX=987654321;

"Iteration:"
IXX=IXX*997353120; "This is Knuth’s 48-bit multiplier times 32"
RNG=0.5+IXX*0.1110223024625157E-15; "0.1110223024625157E-15 is 2**(-53)"
IXX=IXX/32; "Shift to the right by 5 bits to align properly"

4 A universal random number generator, state-of-the-art since 1988

A new “universal” lagged-Fibonacci pseudo random number generator has been developed by Marsaglia, Zaman and Tsang [6, 7]. It will provide identical sequences on all machines that support single-precision real numbers with 24-bit fractional parts. The sequence length is $2^{144}$ (about $2 \times 10^{43}$), long enough for any practical calculation. The MORTRAN3 source code is:
"Declarations:
REAL*4 U(97),C,CD,CM;INTEGER IXX,JXX;

"Initialisation:
IF((IXX.LE.0).OR.(IXX.GT.31328)) IXX=1802; "SETS MARSAGLIA DEFAULT"
IF((JXX.LE.0).OR.(JXX.GT.30081)) JXX=9373; "SETS MARSAGLIA DEFAULT"
I = MOD(IXX/177,177) + 2;
J = MOD(IXX, 177) + 2;
K = MOD(JXX/169,169) + 1;
L = MOD(JXX, 169) ;
DO II=1,97[
   S=0.0;T=0.5;
   DO JJ=1,24[
      M=MOD(MOD(I*J,179)*K,179);
      I=J;J=K;K=M;L=MOD(B3*L+1,169);
      IF(MOD(L*M,64).GE.32) S=S+T;
      T=0.5*T;
   ]
   U(IJ)=S;
]
C = 362436./16777216.;
CD = 7654321./16777216.;
CM = 16777213./16777216.;
IXX = 97;JXX = 33;

"Iteration:"
RNG=U(IXX)-U(JXX); IF(RNG.LT.0.) RNG=RNG+1.; U(IXX) = RNG;
IXX=IXX-1; IF(IXX.EQ.0) IXX=97;
JXX=JXX-1; IF(JXX.EQ.0) JXX=97;
C=C-CD; IF(C.LT.0.) C=C+CM;
RNG=RNG-C; IF(RNG.LT.0.) RNG=RNG+1.;

The salient features to note are:

The declarations define 100 real single-precision numbers and 2 integers. The "state" of this RNG is defined as the state of all these variables. Therefore, if you wish to restart a Monte Carlo run at some point you have to have stored the entire state of the RNG, not just one or two integer seeds. This is a minor price to pay for such a powerful RNG.

The initialisation code is best put into a separate subroutine. This initialisation is performed only once. It specifies the initial state of the RNG, all 102 numbers, based upon the two integer seeds, IXX and JXX. If a Monte Carlo run is restarted, there is no need to repeat the initialisation. IXX and JXX are restricted to the ranges 1 ≤ IXX ≤ 31328 and 1 ≤ JXX ≤ 30081. Unique pairs of IXX and JXX produce independent random number sequences. Thus, it is conceivable that one could run independent Monte Carlo runs on 31,328×30,081 = 942,377,568 computers and combine results at the end with the guarantee that each computer's result is independent of any other's. Besides the intriguing scenario of employing every computer in the world to perform one's application this feature could be employed in parallel or distributed
computing environments or enable a joint Monte Carlo project to be carried out among 2 or
more institutions, without the danger that simulations have been duplicated.

Note that the iteration loop involves no multiplications. (Modern architectures multiply
and add equally fast, usually in one clock cycle.) Each iteration requires 3–6 real additions
or subtractions, 2 integer subtractions, 1–3 assignments. Moreover, there is a great deal of
independence among these instructions allowing compilers to make advantage of “pipelining”
techniques for increased speed. On a typical computer this RNG takes only about 50% longer
than the fastest MCRNG. Clearly, lagged-Fibonacci RNG’s represent the future of random
number generation.

5 Doing it in a UNIX environment

To obtain EGS for use in a UNIX environment, follow the instructions given in the
lecture notes, “How to manage the EGS4 system in a UNIX environment”.

6 Doing it on a PC

This section describes how you would get EGS4 started on a PC. Much of the bootstrapping
procedure is analogous for other machines.

Steps for installation of EGS4 on a PC

1. The easiest way to obtain the PC/DOS EGS4 System (free) is via the Lawrence H. Lanzl
   Institute of Medical Physics www home page at:

   http://sequoia.lanzl.com

   You will find a button to click on with the option to obtain the EGS4 code for PC and
to order the SLAC-265 EGS4 User Manual.

   The PC/DOS EGS4 System can also be obtained by writing to:

   Suzan Walker
   The Lawrence H. Lanzl Institute of Medical Physics
   3600 - 15th Ave. W., Suite 205
   Seattle, WA 98119
   USA
   phone: (206) 286-0241
   fax: (206) 286-0231
   email: susan.walker@lanzl.com

   A nominal fee will be charged for supplies and postage. The P.C. distribution is available
for either 3.25 (1.2MB) or 3.5 (1.44MB) floppy disks. A form for registering as an EGS4
user and for ordering a free copy of the EGS4 User Manual (SLAC-265) with W.R. Nelson
will arrive with the distribution if you have ordered it by mail. Ask for a Mortran3 manual
as well.

   The PC version is also available in archive format (EGS4ARC.EXE) via anonymous ftp at:
   nrcnet0.nrc.ca (132.246.160.2)
cd to pub/egs4 and browse for what you need. The PC version is in subdirectory
PC_Version. Be sure to download both EGS4ARC.EXE and READMEOR.DIE.

If you have received this by anonymous ftp you should send an e-mail request for the
SLAC-265 manual to W.R. Nelson at:
wrnrp@slacvm.slac.stanford.edu

stating that you received the PC version of EGS4 by anonymous ftp, you would like to
register as an EGS4 user and the type of computer you intend to install EGS4 on.

2. To install the EGS4 system on your hard disk, you should have an EGS4ARC.EXE file and
an instruction file called READMEOR.DIE. The extract procedure will create and place files
in the following sample directory structure:

D:\HENHOUSE\EGS4 -------- EGS4 system and batch procedures
D:\HENHOUSE\EGS4\APPENDIX - text versions of SLAC-265 appendices
D:\HENHOUSE\EGS4\BENCHMARK - timing benchmark code XYZDOS
D:\HENHOUSE\EGS4\EXAMPLES - user code examples from SLAC
D:\HENHOUSE\EGS4\EXAMIN --- code for viewing PEGS4 data sets
D:\HENHOUSE\EGS4\INHOM ---- advanced user code example
D:\HENHOUSE\EGS4\INHOMP ---- same as above with PRESTA
D:\HENHOUSE\EGS4\DOSRZ ---- advanced user code example
D:\HENHOUSE\EGS4\ESPECT --- Energy spectrum input files
D:\HENHOUSE\EGS4\TUTOR ---- tutorial codes
D:\HENHOUSE\MORTRAN3 ------ MORTRAN3 pre-compiler and batch procedures
D:\HENHOUSE\PEGS4 -------- PEGS4 cross-section data code and batch procedures
D:\HENHOUSE\PEGS4\DAT ----- home of cross-section data files
D:\HENHOUSE\PEGS4\ESP ----- density-effect correction files

3. IMPORTANT! You must then edit the file EGS4ENV.BAT in the EGS4 subdirectory of your
newly created EGS4 system (e.g. D:\HENHOUSE\EGS4\EGS4ENV.BAT). Edit the line which
sets the default root directory for the EGS4 distribution (the HENHOUSE environment
variable) to reflect your setup. (e.g. change C:\HENHOUSE to D: if you are installing EGS4
into the root directory of drive D:). Also you may choose to edit the lines which set the
path for your Fortran compiler and linker.

All of the EGS4, MORTRAN3, and PEGS4 batch procedures make use of the Lahey FOR-
TRAN and Lahey/Phar Lap 386—DOS Extender FORTRAN compile, link, debug and
execute procedures which are f7713, 386link, sold3, and <filename>.exe respectively.
These commands can be easily adapted for other FORTRAN compiler configurations via
environment variables (Efcompile, Efcompile_dbg, Elink, etc.) which are also set in the
batch file EGS4ENV.BAT.

If you are using a different compiler or linker you will need to change these environment
variables to the appropriate FORTRAN compile and link commands for your system.

Each time you prepare to create or run EGS4 codes you must run the EGS4ENV.BAT file.
If this is frequent you will probably want to put this command in your autoexec.bat file.
Depending on your EGS4 home directory your command would be e.g. f:\egs4\egs4env
or d:\henhouse\egs4\egs4env.
4. FORTRAN compile the \MORTRAN3\MORTRAN3.FOR source code employing the batch command \MORTRAN3\MAKEMOR3.BAT.

5. Bootstrap the MORTRAN3 module by running \MORTRAN3\RAWTOHEX.BAT. This step produces a hexadecimal data file, ..\MORTRAN3\MORTRAN3.DAT, which looks like:

```
v....2C USER F77 11JUN85
00002884000013D9000000000001B6F10001B6F20001B6F1000179E2000179E3000000005
.
.
.
```

6. Delete the first line of non-hex data and save the file as \MORTRAN3\MORTRAN3.DAT.

7. Using the batch procedure %\HENHOUSE%\EGS4\MF.BAT and the MYCONF. BAT in the %\HENHOUSE%\EGS4\TUTOR subdirectory, attempt to MORTRAN, FORTRAN, LINK some of the \EGS4\TUTOR\TUTOR#.MOR codes. Sample command for this step which must be from the TUTOR subdirectory is: D:\HENHOUSE\EGS4\MF TUTOR1

8. Build the PEGS4 executable module by utilizing the batch command %\HENHOUSE%\PEGS4\MAKEPEGS.BAT. Following the examples of PEGS4.INP files provided and the information on using PEGS4 in the SLAC-265 User Manual, create data sets needed for the TUTOR#.MOR codes. (Note: the files HI.DAT, 700.DAT, 521.DAT and WATER.DAT provided in the distribution can be used with the TUTOR codes.)

9. Run the TUTOR# codes using the %\HENHOUSE%\EGS4\EX.BAT batch command, where your command line, for example, from C:\EGS4\TUTOR\ would be:

```
D:\HENHOUSE\EGS4\EX TUTOR1 NULL HI
```

(NULL being empty input file for TUTOR code cases).

10. Create your own user codes using the TUTOR#, XYZDOS, DOSRZ, or INHOM user codes as starting points.

11. If you are interested in subscribing to the EGS4 listserv account to post and receive questions and answers regarding the use of EGS4 you can do so by the following:

To sign on, send an e-mail message to:
listerv@slacvm.slac.stanford.edu

saying:
SUBscribe EGS4-L "your full name"

To post questions/comments/answers, send e-mail to:
egs4-l@slacvm.slac.stanford.edu

To sign off, send an e-mail message to:
listerv@slacvm.slac.stanford.edu

saying:
UNSUB EGS4-L

Congratulations and welcome to EGS4!
References


