SPEC CPU2006 Benchmark Descriptions

Descriptions written by the SPEC CPU Subcommittee and by the original program authors [1].
Edited by John L. Henning,
Secretary, SPEC CPU Subcommittee, and Performance Engineer, Sun Microsystems.
Contact john.henning@acm.org

Introduction

On August 24, 2006, the Standard Performance Evaluation Corporation (SPEC) announced CPU2006 [2], which replaces CPU2000. The SPEC CPU benchmarks are widely used in both industry and academia [3].

The new suite is much larger than the previous, and will exercise new corners of CPUs, memory systems, and compilers – especially C++ compilers. Where CPU2000 had only 1 benchmark in C++, the new suite has 7, including one with ½ million lines of C++ code. As in previous CPU suites, Fortran and C are also well represented.

Since its beginning, SPEC has claimed the motto that

"An ounce of honest data is worth a pound of marketing hype".

To help keep the benchmarking data honest, fair, and relevant, SPEC CPU draws benchmarks from real life applications, rather than using artificial loop kernels or synthetic benchmarks. Therefore, the most important parts of the new suite are the benchmarks themselves, which are described on the pages that follow. In a future issue of Computer Architecture News, information will be provided about other aspects of the new suite, including additional technical detail regarding benchmark behavior and profiles.

References:

[1] Program authors are listed in the descriptions below, which are adapted from longer versions posted at www.spec.org/cpu2006/Docs/. The SPEC project leaders are listed in credits.html at the same location.
[2] SPEC's press announcement may be found at www.spec.org/cpu2006/press/release.html
[3] SPEC's website has over 6000 published results for CPU2000, at www.spec.org/cpu2000/results. Google reports 270,000 hits for the phrase "SPEC CPU2000" as of September 2006.

The benchmarks are described in order by category - first the integer benchmarks, then the floating point benchmarks.

Part 1: Integer Benchmarks

- 400.perlbench
- 401.bzip2
- 403.gcc
- 429.mcf
- 445.gobmk
- 456.hmmer
- 458.sjeng
- 462.libquantum
- 464.h264ref
- 471.omnetpp
- 473.astar
- 483.xalancbmk

Part 2: Floating Point Benchmarks

- 410.bwaves
- 416.gamess
- 433.milc
- 434.zeusmp
- 435.gromacs
- 436.cactusADM
- 437.leslie3d
- 444.namd
- 447.dealII
- 450.soplex
- 453.povray
- 454.calculix
- 459.GemsFDTD
- 465.tonto
- 470.lbm
- 481.wrf
- 482.sphinx3
- 999.specrand

.

Part 1: Integer Benchmarks

400.perlbench

Authors: Larry Wall, et. al.

General Category: Programming language

Description: 400.perlbench is a cut-down version of Perl v5.8.7, the popular scripting language. SPEC's version of Perl has had most of OS-specific features removed. In addition to the core Perl interpreter, several third-party modules are used:

- SpamAssassin v2.61
- Digest-MD5 v2.33
- HTML-Parser v3.35
- MHonArc v2.6.8
- IO-stringy v1.205
- MailTools v1.60
- TimeDate v1.16

Sources for all of the freely-available components used in 400.perlbench can be found on the distribution media in the original.src directory.

Input: The reference workload for 400.perlbench consists of three scripts:

1. The primary component of the workload is the Open Source spam checking software SpamAssassin. SpamAssassin is used to score a couple of known corpora of both spam and ham (non-spam), as well as a sampling of mail generated from a set of random components. SpamAssassin has been heavily patched to avoid doing file I/O, and does not use the Bayesian filtering.

2. Another component is the popular freeware email-to-HTML converter MHonArc. Email messages are generated randomly and converted to HTML. In addition to MHonArc, which was lightly patched to avoid file I/O, this component also uses several standard modules from the CPAN (Comprehensive Perl Archive Network).

3. The third script in the reference workload (which also uses the mail generator for convienience) excercises a slightly-modified version of the 'specdiff' script, which is a part of the CPU2006 tool suite.

The training workload is similar, but not identical, to the reference workload from CPU2000.

The test workload consists of the non-system-specific parts of the actual Perl 5.8.7 test harness.

Output: In the case of the mail-based benchmarks, a line with salient characteristics (number of header lines, number of body lines, etc) is output for each message generated. During processing, MD5 hashes of the contents of output "files" (in memory) are computed and output. For SpamAssassin, the message's score and the rules that it triggered are also output.

Programming Language: ANSI C

Known Portability Issues: There are some known aliasing issues. The internal data structures that represent Perl's variables are accessed in such as a way as to violate ANSI aliasing rules. Compilation with optimizations that rely on strict compliance to ANSI C aliasing rules will most likely produce binaries that will not validate.

References:

[1] Perl Mongers: http://www.perl.org/
[2] O'Reilly's Perl Pages: http://www.perl.com/
[3] The Comprehensive Perl Archive Network: http://www.cpan.org/
[4] SpamAssassin: http://spamassassin.apache.org/
[5] MHonArc: http://www.mhonarc.org/

401.bzip2 *(Integer benchmarks cont'd)*

Author: Julian Seward

General Category: Compression

Description: 401.bzip2 is based on Julian Seward's bzip2 version 1.0.3.

The only difference between bzip2 1.0.3 and 401.bzip2 is that SPEC's version of bzip2 performs no file I/O other than reading the input. All compression and decompression happens entirely in memory. This is to help isolate the work done to only the CPU and memory subsystem.

Input: 401.bzip2's reference workload has six components: two small JPEG images, a program binary, some program source code in a tar file, an HTML file, and a "combined" file, which is representative of an archive that contains both highly compressible and not very compressible files.

Each input set is compressed and decompressed at three different blocking factors ("compression levels"), with the end result of the process being compared to the original data after each decompression step.

Output: The output files provide a brief outline of what the benchmark is doing as it runs. Output sizes for each compression and decompression are printed to facilitate validation, and the results of decompression are compared with the input data to ensure that they match.

Programming Language: ANSI C

Known Portability Issues: None

References:

[1] Michael Burrows and D. J. Wheeler: "A block-sorting lossless data compression algorithm" 10th May 1994. Digital SRC Research Report 124. ftp://ftp.digital.com/pub/DEC/SRC/research-reports/SRC-124.ps.gz

[2] Daniel S. Hirschberg and Debra A. LeLewer, "Efficient Decoding of Prefix Codes", Communications of the ACM, April 1990, Vol 33, # 4.

[3] David J. Wheeler, Program bred3.c and accompanying document bred3.ps. ftp://ftp.cl.cam.ac.uk/users/djw3/

[4] Jon L. Bentley and Robert Sedgewick, "Fast Algorithms for Sorting and Searching Strings", Available from Sedgewick's web page, http://www.cs.princeton.edu/~rs/

[5] Peter Fenwick, "Block Sorting Text Compression -- Final Report", The University of Auckland, Department of Computer Science Report No. 130, April 1996. ftp://ftp.cs.auckland.ac.nz/pub/staff/peter-f/TechRep130.ps

403.gcc *(Integer benchmarks cont'd)*

Author: Richard Stallman and a large cast of helpers [1].

General Category: C Language optimizing compiler

Description: 403.gcc is based on gcc Version 3.2. It generates code for an AMD Opteron processor. The benchmark runs as a compiler with many of its optimization flags enabled.

403.gcc has had its inlining heuristics altered slightly, so as to inline more code than would be typical on a Unix system in 2002. It is expected that this effect will be more typical of compiler usage in 2006. This was done so that 403.gcc would spend more time analyzing its source code inputs, and use more memory. Without this effect, 403.gcc would have done less analysis, and needed more input workloads to achieve the run times required for CPU2006.

Input: There are 9 input workloads in 403.gcc. These files are preprocessed C code (.i files):

- cp-decl.i and expr.i come from the source files of 176.gcc from CPU2000.
- 166.i is made by concatenating the Fortran source files of a SPECint2000 candidate benchmark, then using the f2c translator to produce C code, and then pre-processing.
- 200.i comes via the same method from a previous version of the SPECfp2000 benchmark 200.sixtrack.
- scilab.i comes via the same method from a version of the Scilab program.
- Expr2.i comes from the source of 403.gcc, as does c-typeck.i.
- g23.i comes from fold-const.c from 403.gcc, and s04.i comes from sched-deps.c of 403.gcc

Output: All output files are x86-64 assembly code files.

Programming Language: C

Known Portability Issues:

- Some of the optimizations 403.gcc performs require constant propagation of floating point constants. These form an insignificant amount of computation time, yet may depend on IEEE floating point format to produce a correct result.
- 403.gcc is not an ANSI C program. It uses GNU extensions.
- The initial port of 403.gcc was to a 64 bit system. It has been successfully ported by SPEC to many 32-bit UNIX implementations.

References:

[1] http://gcc.gnu.org/onlinedocs/gcc/Contributors.html .

[2] See the GCC home page at http://gcc.gnu.org

429.mcf *(Integer benchmarks cont'd)*

Author: Andreas Löbel; SPEC Project Leader: Reinhold Weicker

General Category: Combinatorial optimization / Single-depot vehicle scheduling

Description: 429.mcf is derived from MCF, a program used for single-depot vehicle scheduling in public mass transportation.

The program is designed for the solution of single-depot vehicle scheduling problems planning transportation. It considers one single depot and a homogeneous vehicle fleet. Based on a line plan and service frequencies, so-called timetabled trips with fixed departure/arrival locations and times are derived. Each of these timetabled trips has to be serviced by exactly one vehicle. The links between these trips are called dead-head trips. In addition, there are pull-out and pull-in trips for leaving and entering the depot.

Cost coefficients are given for all dead-head, pull-out, and pull-in trips. It is the task to schedule all timetabled trips to so-called blocks such that the number of necessary vehicles is as small as possible and, subordinate, the operational costs among all minimal fleet solutions are minimized.

For the considered single-depot case, the problem can be formulated as a large-scale minimum-cost flow problem, solved with a network simplex algorithm accelerated with a column generation.

The network simplex algorithm is a specialized version of the well known simplex algorithm for network flow problems. The linear algebra of the general algorithm is replaced by simple network operations such as finding cycles or modifying spanning trees that can be performed very quickly. The main work of our network simplex implementation is pointer and integer arithmetic.

In the transition from 181.mcf (CPU2000) to 429.mcf (CPU2006), new inputs were defined for test, train, and ref, with the goal of longer execution times. The heap data size, and with it the overall memory footprint, increased accordingly. Most of the source code was not changed, but. several type definitions were changed by the author:

- Whenever possible, long typed attributes of struct node and struct arc are replaced by 32 bit integer, for example if used as boolean type. Pointers remain unaffected and map to 32 or 64 bit long, depending on the compilation model, to ensure compatibility to 64 bit systems.
- To reduce cache misses and accelerate program performance somewhat, the elements of struct node and struct arc, respectively, are rearranged according to the proposals made in [1].

Input: The input file contains: the number of timetabled and dead-head trips; for each timetabled trip its starting and ending time; for each dead-head trip its starting and ending timetabled trip and its cost.

Worst case execution time is pseudo-polynomial in the number timetabled and dead-head trips and in the amount of the maximal cost coefficient. The expected execution time, however, is in the order of a low-order polynomial.

Memory Requirements: 429.mcf requires about 860 and 1700 megabyte for a 32 and a 64 bit data model, respectively.

Output: The benchmark writes log information, a checksum, and output values describing an optimal schedule.

Programming Language: ANSI C, mathematical library (libm) required.

Known Portability Issues: In the SPEC version, -DWANT_STDC_PROTO is set to require ANSI C prototyping, for reasons of compatibility and standards.

References:

[1] Marty Itzkowitz, Brian Wylie, Christopher Aoki, and Nicolai Kosche, "Memory Profiling using Hardware Counters" www.sc-conference.org/sc2003/paperpdfs/pap182.pdf

[2] Background on the vehicle scheduling problem can be found in the author's Ph.D. thesis "Optimal Vehicle scheduling in public transit", www.zib.de/loebel or at ftp://ftp.zib.de/pub/zib-publications/books/Loebel.disser.ps

[3] The work horse in the benchmark 429.mcf is the code MCF Version 1.2, which is free for academic use at www.zib.de/Optimization/Software/Mcf/ Compared with the original and the commercial versions, the benchmark version has simplified I/O. The main algorithmic part, however, has been retained.

[4] An excellent text book about the network simplex algorithm and network flow in general is Ahuja, Magnanti, and Orlin: "Network Flows: Theory, Algorithms, and Applications", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1993.

[5] MCF had originally been developed for application in the public transportation systems of Hamburg and Berlin (BVG). For BVG, bus scheduling was optimized in 1998 on the basis of MCF; BVG also owns usage rights to the software that has been integrated into their planning system BERTA.

[6] The MCF method for vehicle scheduling later has been integrated, into the vehicle and personel planning system MICROBUS. This system in now marketed by IVU Traffic Technologies AG (http://www.ivu.de); the bus service divisions of the German and the Austrian railway companies are among the licencees.

445.gobmk *(Integer benchmarks cont'd)*

GNU Go authors: (in chronological order of contribution) are Man Lung Li, Wayne Iba, Daniel Bump, David Denholm, Gunnar Farnebäck, Nils Lohner, Jerome Dumonteil, Tommy Thorn, Nicklas Ekstrand, Inge Wallin, Thomas Traber, Douglas Ridgway, Teun Burgers, Tanguy Urvoy, Thien-Thi Nguyen, Heikki Levanto, Mark Vytlacil, Adriaan van Kessel, Wolfgang Manner, Jens Yllman, Don Dailey, Mans Ullerstam, Arend Bayer, Trevor Morris, Evan Berggren Daniel, Fernando Portela, Paul Pogonyshev, S.P. Lee, Stephane Nicolet and Martin Holters.

General Category: Artificial intelligence - game playing.

Description: The program plays Go and executes a set of commands to analyze Go positions.

Input: Most input is in "SmartGo Format" (.sgf), a widely used de facto standard representation of Go games. A typical test involves reading in a game to a certain point, then executing a command to analyze the position.

Output: typically an ascii description of a sequence of Go moves.

Programming Language: C

Known Portability Issues: There are no known portability problems remaining. The last portability problem fixed dealt was the nonstandard assumption of signed characters.

References:

[1] www.gnu.org/software/gnugo/devel.html

456.hmmer *(Integer benchmarks cont'd)*

Author: Sean Eddy; SPEC Project Leader: Kaivalya M. Dixit (IBM), Alan MacKay (IBM)

General Category: Search a gene sequence database

Description: Profile Hidden Markov Models (profile HMMs) are statistical models of multiple sequence alignments, which are used in computational biology to search for patterns in DNA sequences.

The technique is used to do sensitive database searching, using statistical descriptions of a sequence family's consensus. It is used for protein sequence analysis.

Input: A database (sprot41.dat) is used in the reference workloads. An input workload (nph3.hmm) is used to find a ranked list of best sorting sequences from the sprot41.dat file using the hmmsearch function.

For test, train, and 1 of the 2 reference workloads, 3 different hmm files are used to with the hmmcalibrate function to calibrate HMM search statistics. This function scores a large number of synthesized random sequences from the input file and fits an extreme value distribution (EVD) to the histogram of those scores.

Output: Four output files contain a ranked list of matches.

Programming Language: C Language

Known Portability Issues: All resolved during porting it for SPEC.

References:

[1] http://hmmer.wustl.edu

458.sjeng *(Integer benchmarks cont'd)*

Authors: Gian-Carlo Pascutto, Vincent Diepeveen

General Category: Artificial Intelligence (game tree search & pattern recognition)

Description: 458.sjeng is based on Sjeng 11.2, which is a program that plays chess and several chess variants, such as drop-chess (similar to Shogi), and 'losing' chess.

It attempts to find the best move via a combination of alpha-beta or priority proof number tree searches, advanced move ordering, positional evaluation and heuristic forward pruning. Practically, it will explore the tree of variations resulting from a given position to a given base depth, extending interesting variations but discarding doubtful or irrelevant ones. From this tree the optimal line of play for both players ("principle variation") is determined, as well as a score reflecting the balance of power between the two.

The SPEC version is an enhanced version of the free Sjeng 11.2 program, modified to be more portable and more accurately reflect the workload of current professional programs.

Input: 458.sjeng's input consists of a textfile containing alternations of a chess position in the standard Forsyth-Edwards Notation (FEN) and the depth to which this position should be analyzed, in half-moves (ply depth)

The SPEC reference input consists of 9 positions belonging to various phases of the game.

Output: 458.sjeng's output consists, per position, of some side information (textual display of the chessboard, phase of the game, used parameters...) followed by the output from the tree searching module as it progresses. This is formatted as: Attained depth in half-moves (plies); Score for the player that is to move, in equivalents of 1 pawn; Number of positions investigated; and the optimal line of play ("principle variation")

Programming Language: ANSI C

Known Portability Issues: Requires that "int" is at least 32 bits wide.

References:

[1] Sjeng 11.2 & Deep Sjeng: www.sjeng.org

[2] Portable Game Notation Specification (including FEN/EPD): ww.tim-mann.org/Standard

462.libquantum *(Integer benchmarks cont'd)*

Author: Björn Butscher, Hendrik Weimer

General Category: Physics / Quantum Computing

Description: libquantum is a library for the simulation of a quantum computer. Quantum computers are based on the principles of quantum mechanics and can solve certain computationally hard tasks in polynomial time.

In 1994, Peter Shor discovered a polynomial-time algorithm for the factorization of numbers, a problem of particular interest for cryptanalysis, as the widely used RSA cryptosystem depends on prime factorization being a problem only to be solvable in exponential time. An implementation of Shor's factorization algorithm is included in libquantum.

Libquantum provides a structure for representing a quantum register and some elementary gates. Measurements can be used to extract information from the system. Additionally, libquantum offers the simulation of decoherence, the most important obstacle in building practical quantum computers. It is thus not only possible to simulate any quantum algorithm, but also to develop quantum error correction algorithms. As libquantum allows to add new gates, it can easily be extended to fit the ongoing research, e.g. it has been deployed to analyze quantum cryptography.

Input: The benchmark program expects the number to be factorized as a command-line parameter. An additional parameter can be supplied to specify a base for the modular exponentiation part of Shor's algorithm.

Output: The program gives a brief explanation on what it is doing and the factors of the input number if the factorization was successful.

Programming Language: ISO/IEC 9899:1999 ("C99")

Known Portability Issues: On Solaris prior to version 10, /usr/include/complex.h is not available, You can work around this by including -I. –lcplxsupp in your compilation flags.

References:

[1] libquantum Website: www.enyo.de/libquantum/

[2] Wikipedia article on Quantum Computer http://en.wikipedia.org/wiki/Quantum_computer

[3] Peter W. Shor: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer www.arxiv.org/abs/quant-ph/9508027

464.h264ref *(Integer benchmarks cont'd)*

Author: Karsten Sühring [1] and many others [2] wrote the H.264/AVC reference implementation. Tom Pycke has coordinated packaging the reference version for use by SPEC.

General Category: Video compression

Description: 464.h264ref is a reference implementation of H.264/AVC (Advanced Video Coding), the latest video compression standard, developed by the VCEG (Video Coding Experts Group) of the ITU [3] and the MPEG [4] group of the ISO/IEC [5]. This standard replaces the current MPEG-2 standard, for applications such as the next-generation DVDs (Blu-ray and HD DVD) and video broadcasting.

464.h264ref is based on version 9.3 of the h264avc reference implementation, modified to improve portability, validation, and fairness. I/O and platform-specific code were reduced. It encodes video using 2 parameter sets: (1) Basic profile (baseline.cfg): Good compression, fast encoding. This profile can be used for real-time encoding applications such as video conferencing. (2)Main profile (main.cfg): Best compression. This can be used in applications where no loss of data can occur such as the next generation DVD.

Input: 2 files in (uncompressed) video data in YUV-format.: Foreman: a standard sequence used in video compression with 120 frames at 176x144 pixels; and SSS, a sequence from a video game, with 171 frames at 512x320 pixels

Output: Included are encode logs from foreman in both baseline and the main profile, and sss in the main profile.

Copyrights and disclaimers: The 464.h264ref benchmarks sources are protected by the original copyright notice that is part of the official h264avc reference software, version 9.3 [6]. The full text of the copyright, and important disclaimers, are reproduced at [7].

Programming Language: C

Known Portability Issues: None

References:

[1] http://iphome.hhi.de/suehring/tml/index.htm
[2] http://iphome.hhi.de/suehring/tml/doc/lenc/html/contributors_8h.html#_details
[3] International Telecommunications Union, www.itu.int
[4] Moving Pictures Experts Group, www.chiariglione.org/mpeg
[5] International Standardization Organization, www.iso.ch.
[6] http://iphome.hhi.de/suehring/tml/download/old_jm/jm39.zip
[7] www.spec.org/cpu2006/docs/464.h264ref.html
[8] http://en.wikipedia.org/wiki/H264

471.omnetpp *(Integer benchmarks cont'd)*

Author: András Varga, Omnest Global, Inc.

General Category: Discrete Event Simulation

Description: simulation of a large Ethernet network, based on the OMNeT++ discrete event simulation system [1], using an ethernet model which is publicly available [2].

For the reference workload, the simulated network models a large Ethernet campus backbone, with several smaller LANs of various sizes hanging off each backbone switch. It contains about 8000 computers and 900 switches and hubs, including Gigabit Ethernet, 100Mb full duplex, 100Mb half duplex, 10Mb UTP, and 10Mb bus. The training workload models a small LAN.

The model is accurate in that the CSMA/CD protocol of Ethernet and the Ethernet frame are faithfully modelled. The host model contains a traffic generator which implements a generic request-response based protocol. (Higher layer protocols are not modelled in detail.)

Input: The topology of the network and structure of hosts, switches and hubs are described in NED files (the network description language of OMNeT++.) Operation of the Ethernet MAC, traffic generator etc. are in C++.

Request and reply lengths are configured as intuniform (50,1400) and truncnormal (5000,5000) for the reference input. The volume of the traffic can most easily be controlled with the time period between sending requests; currently set in omnetpp.ini to exponential(0.33) (that is, average 3 requests per second) for the reference input. This already causes frames to be dropped in some of the backbone switches, so the network is a bit overloaded with the current settings.

Output: The model generates extensive statistics in the omnetpp.sca file at the end of the simulation: number of frames sent, received, dropped, etc. These are only basic statistics; however, if all nodes were allowed to record them, omnetpp.sca would grow to about 28 megabytes. To make the output more reasonable in size, recording statistics is only enabled in a few nodes.

Programming Language: C++

Known Portability Issues: None.

References:

[1] OMNeT++ Community Site: www.omnetpp.org
[2] Ethernet simulation model: http://ctieware.eng.monash.edu.au/twiki/bin/view/Simulation/EtherNet
[3] Ethernet model documentation: http://ctieware.eng.monash.edu.au/~ctieware/ethernet-doc

473.astar *(Integer benchmarks cont'd)*

Author: Lev Dymchenko

General Category: Computer games. Artificial Intelligence. Path finding.

Description: 471.astar (pronounced: A-star) is derived from a portable 2D path-finding library that is used in game's AI. This library implements three different path-finding algorithms:

First is the well known A* algorithm for maps with passable and non-passable terrain types.

Second is a modification of the A* path finding algorithm for maps with different terrain types and different move speed.

Third is an implementation of A* algorithm for graphs. This is formed by map regions with neighborhood relationship.

The library also includes pseudo-intellectual functions for map region determination.

Input: The input file is a map in binary format. The program also accepts typical map region size which is used in region-based path finding algorithm and density for randomly created forest-style test maps. The program also reads the number of ways to simulate.

Output: The program outputs the number of existing ways and the total way length to validate correctness.

Programming Language: C++

Known Portability Issues: None

References:

[1] www.gamasutra.com/features/19990212/sm_01.htm Popular article about A* algorithm.

[2] www.policyalmanac.org/games/aStarTutorial.htm a tutorial on A* algorithm.

483.xalancbmk *(Integer benchmarks cont'd)*

Author: IBM Corporation, Apache Inc, plus modifications for SPEC purposes by Christopher Cambly, Andrew Godbout, Neil Graham, Sasha Kasapinovic, Jim McInnes, June Ng, Michael Wong. Primary contact: Michael Wong

General Category: XSLT processor for transforming XML documents into HTML, text, or other XML document types

Description: a modified version of Xalan-C++ [1], an XSLT processor written in a portable subset of C++ . Xalan-C++ version 1.8 is a robust implementation of the W3C Recommendations for XSL Transformations (XSLT) [2] and the XML Path Language (XPath) [3]. It works with a compatible release of the Xerces-C++ [4] XML parser: Xerces-C++ version 2.5.0. The XSLT language is use to compose XSL stylesheets. An XSL stylesheet contains instructions for transforming XML documents from one document type to another document type (XML, HTML, or other). In structural terms, an XSL stylesheet specifies the transformation of one tree of nodes (the XML input) into another tree of nodes (the output or transformation result).

Modifications for SPEC benchmarking purposes include: combining code to make a standalone executable, removing compiler incompatibilities and improving standard conformance, changing output to display intermediate values, removing large parts of unexecuted code, and moving all the include locations to fit better into the SPEC harness.

Input: An XML document and an XSL Stylesheet.

Output: An HTML document

Programming Language: C++

Known Portability Issues: None

References:

[1] http://xml.apache.org/xalan-c/

[2] Xalan-C++ fully implements the W3C Recommendation 16 November 1999 XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/xslt

[3] Xalan-C++ incorporates the XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath.

[4] Xalan-C++ uses Xerces-C++ to parse XML documents and XSL stylesheets: http://xml.apache.org/xerces-c.

[5] Xalan-C++ supports C++ extension functions http://xml.apache.org/xalan-c/extensions.html .

Part 2: Floating Point Benchmarks

410.bwaves

Author: Dr. Mark Kremenetsky, Silicon Graphics, Inc

General Category: Computational Fluid Dynamics

Description: 410.bwaves numerically simulates blast waves in three dimensional transonic transient laminar viscous flow.

The initial configuration of the blast waves problem consists of a high pressure and density region at the center of a cubic cell of a periodic lattice, with low pressure and density elsewhere. Periodic boundary conditions are applied to the array of cubic cells forming an infinite network. Initially, the high pressure volume begins to expand in the radial direction as classical shock waves. At the same time, the expansion waves move to fill the void at the center of the cubic cell. When the expanding flow reaches the boundaries, it collides with its periodic images from other cells, thus creating a complex structure of interfering nonlinear waves. These processes create a nonlinear damped periodic system with energy being dissipated in time. Finally, the system will come to an equilibrium and steady state.

The algorithm implemented is an unfactored solver for the implicit solution of the compressible Navier-Stokes equations using the Bi-CGstab algorithm, which solves systems of non-symmetric linear equations iteratively.

Input: describes the grid size, flow parameters, initial boundary condition and number of time steps. The test, train and ref data sets differ only in grid size and number of steps.

Output: The transient nature of the flow and iterative solver makes bwaves a difficult problem to validate. In SPEC CPU2006 this has been addressed by comparing three different outputs. These are: the L2 norm of dq(l.i.j.k) vector after final time step; the residual for convergence after each time step; and (3) the cumulative sum of iterations for convergence for every time step

Programming Language: Fortran 77

Known Portability Issues: none

416.gamess *(Floating point benchmarks cont'd)*

Author: Gordon Research Group, Iowa State University [1]

General Category: Quantum chemical computations

Description: A wide range of quantum chemical computations are possible using GAMESS. The benchmark 416.gamess does the following computations for the reference workload: (1) Self-consistent field (SCF) computation (type: Restricted Hartree-Fock) of cytosine molecule using the direct SCF method; (2) SCF computation (type: Restricted open-shell Hartee-Fock) of water and cu2+ using the direct SCF method; (3) SCF computation (type: Multi-configuration Self-consisted field) of triazolium ion using the direct SCF method

Inputs and Outputs: Described in the benchmark Docs subdirectory files INPUT.TXT, INTRO.TXT and PROG.TXT

Programming Language: Fortran

Known Portability Issues: Uses features that are either non-standard or deleted in Fortran 95: equivalence of objects of character type with non-character types; some formal and actual arguments of subroutines do not have the same data type; some arrays are accessed past the end of the defined array size; argument array sizes defined in some subroutines do not match the size of the actual argument passed.

The benchmark source is generated from the original GAMESS by using the 'SEQ' (sequential) directive. So, 416.gamess can not be used for a parallel run.

References:

[1] www.msg.ameslab.gov/GAMESS/GAMESS.html
[2] GAMESS, M.W.Schmidt, K.K.Baldridge, J.A.Boatz, S.T.Elbert, M.S.Gordon, J.J.Jensen, S.Koseki, N.Matsunaga, K.A.Nguyen, S.Su, T.L.Windus, M.Dupuis, J.A.Montgomery J.Comput.Chem. 14, 1347-1363 (1993)
[3] For more references, please see REFS.TXT in the Docs/ subdirectory of the benchmark tree.

433.milc *(Floating point benchmarks cont'd)*

Author: Submitted by Steven Gottlieb for the MILC collaboration

General Category: Physics / Quantum Chromodynamics (QCD)

Description: MILC is developed by the MIMD Lattice Computation (MILC) collaboration for doing simulations of four dimensional SU(3) lattice gauge theory on MIMD parallel machines. The code is used for millions of node hours at DOE and NSF supercomputer centers.

433.milc in SPEC CPU2006 uses the serial version of the su3imp program, which is important and relevant because parallel performance depends on good single processor performance.

The program generates a gauge field, and is used in lattice gauge theory applications involving dynamical quarks. Lattice gauge theory involves the study of some of the fundamental constituents of matter, namely quarks and gluons. In this area of quantum field theory, traditional perturbative expansions are not useful. Introducing a discrete lattice of space-time points is the method of choice.

Input and Outputs: are described with comments at www.spec.org/cpu2006/Docs/433.milc.html. .

Output: Non-timing sections of output is used to verify correctness.

Programming Language: C

References:

[1] http://physics.indiana.edu/~sg
[2] http://physics.indiana.edu/~sg/milc.html

434.zeusmp *(Floating point benchmarks cont'd)*

Author: Michael Norman, University of California, San Diego

General Category: Physics / Magnetohydrodynamics

Description: based on ZEUS-MP, a computational fluid dynamics code developed at the Laboratory for Computational Astrophysics (NCSA, University of Illinois at Urbana-Champaign) for the simulation of astrophysical phenomena.

The program solves equations of ideal, non-relativistic, hydrodynamics and magnetohydrodynamics, including gravitational fields. The physical problem solved in SPEC CPU2006 is a 3-D blastwave simulated with the presence of a uniform magnetic field along the x-direction. A Cartesian grid is used and the boundaries are "outflow."

Input: The input file is zmp_inp, which is described at www.spec.org/cpu2006/Docs/434.zeusmp.html. The file includes information on physical constants, grid information, information on Equation of State, and problem control information. In this test, it is a spherical blastwave.

Output: The output file contains physical information of the blastwave at the beginning and the end of the run.

Programming Language: The main code is written in Fortran 77, with the change that (as in all of SPEC's Fortran benchmarks) DOUBLE PRECISION has been replaced by REAL*8

Known Portability Issues: None

References:

[1] http://cosmos.ucsd.edu/lca-www/software/lca_intro_zeusmp.html

435.gromacs *(Floating point benchmarks cont'd)*

Author: Erik Lindahl, Stockholm Bioinformatics Centre; David van der Spoel, Uppsala University

General Category: Chemistry / Molecular Dynamics

Description: 435.gromacs is derived from GROMACS, a versatile package that performs molecular dynamics, i.e. simulation of the Newtonian equations of motion for systems with hundreds to millions of particles. Although primarily designed for biochemical molecules such as proteins and lipids that have many complicated bonded interactions, GROMACS is also fast at calculating the nonbonded interactions that usually dominate the simulation cost. Therefore,it is also used for research on non-biological systems, such as polymers.

The benchmark version performs a simulation of the protein Lysozyme in a solution of water and ions. The structure of a protein is normally determined by experimental techniques such as X-ray crystallography of NMR spectroscopy. By simulating the atomic motions of these structures, one can gain significant understanding of protein dynamics and function, and, in some cases, it might even be possible to predict the structure of new proteins.

A dodecahedron-shaped box is used to reduce the amount of solvent water, but there are still 23179 atoms in the system. The simulation time is dominated by inner loops where nonbonded Lennard-Jones and Coulomb interactions are calculated between atoms closer than 1.4 nm in space.

Input: gromacs.tpr with identical setup but a different number of steps: 1500 for test, 3000 for train, and 6000 for ref

Output: average potential energy and system temperature, and the number of floating point operations performed. Molecular dynamics is a chaotic process. Normally, quite agressive compiler optimizations are used to compile the code, hence slight variations are normal. The results shouldn't differ by more than 1.25% from the reference values.

Programming Language: C and Fortran. The only Fortran code is the inner loops (innerf.f).which typically account for more than 95% of the runtime.

Known Portability Issues: If a Fortran compiler does not append an underscore to external names, you can use the flag -DSPEC_CPU_APPEND_UNDERSCORE

SPEC has chosen some settings in config.h to select whether the C compiler supports strdup(), strcasecmp(), etc. If a compiler should come along that is incompatible with SPEC's settings in config.h, SPEC should be notified.

References:

[1] http://www.gromacs.org

436.cactusADM *(Floating point benchmarks cont'd)*

Author: Malcolm Tobias, Washington University School of Medicine

General Category: Physics / General Relativity

Description: CactusADM is a combination of Cactus, an open source problem solving environment, and BenchADM, a computational kernel representative of many applications in numerical relativity (ADM stands for ADM formalism developed by Arnowitt, Deser and Misner). CactusADM solves the Einstein evolution equations, which describe how spacetime curves as response to its matter content, and are a set of ten coupled nonlinear partial differential equations, in their standard ADM 3+1 formulation. A staggered-leapfrog numerical method is used to carry out the update.

Input: The program requires a parameter file BenchADM.par. This file defines the grid size, as well as the number of iterations which the code will run. The input file can be modified to print out timing information, but this has not been implemented in CPU2006. The initial data represents flat space. Geodesic slicing is used.

Output: The iteration, time, and gxx and gyz components of the metric which are coordinate-dependent descriptions of the space time are printed for validation.

Programming Language: Fortran 90, ANSI C

Known Portability Issues: None

References:

[1] The Cactus Code web site: www.cactuscode.org

437.leslie3d *(Floating point benchmarks cont'd)*

Authors: Christopher Stone, Suresh Menon Georgia Institute of Technology

General Category: Computational Fluid Dynamics (CFD)

Description: 437.leslie3d is derived from LESlie3d (*L*arge-*E*ddy *S*imulations with *L*inear-*E*ddy Model in *3D*), a research-level Computational Fluid Dynamics (CFD) code used to investigate a wide array of turbulence phenomena such as mixing, combustion, and acoustics.

LESlie3d uses a strongly-conservative, finite-volume algorithm with the MacCormack Predictor-Corrector time integration scheme. The accuracy is fourth-order spatially and second-order temporally.

For CPU2006, the program solves a test problem using the temporal mixing layer. This type of flow occurs in the mixing regions of all combustors that employ fuel injection (which is nearly all combustors). The benchmark version, 437.leslie3d, performs limited file I/O using a theoretically exact problem.

Input: Three different input stack sizes, test, train, ref, are available, representing an increasing grid resolution for the solution. Input parameters include the grid size, flow parameters and boundary conditions.

Output: The output includes analysis information that tracks the momentum thickness through time.

Programming Language: Fortran 90

Known Portability Issues: None known.

References:

[1] www.ccl.gatech.edu

444.namd *(Floating point benchmarks cont'd)*

Author: Jim Phillips, University of Illinois

General Category: Scientific, Structural Biology, Classical Molecular Dynamics Simulation

Description: The 444.namd benchmark is derived from the data layout and inner loop of NAMD, a parallel program for the simulation of large biomolecular systems.

Although NAMD was a winner of a 2002 Gordon Bell award for parallel scalability, serial performance is equally important to the over 10,000 users who have downloaded the program over the past several years. Almost all of the runtime is spent calculating inter-atomic interactions in a small set of functions. This set was separated from the bulk of the code to form a compact benchmark for CPU2006. This computational core achieves good performance on a wide range of machines, but contains no platform-specific optimizations.

Input: a 92224 atom simulation of apolipoprotein A-I is used as a standard NAMD benchmark. This particular file format is created by NAMD 2.5 using the "dumpbench" command, and eliminates the need for file readers and other setup code from the benchmark. Test, train and ref read from the same input file, but run the code for different number of iterations. For ref the code is run for 38 iterations.

Output: various checksums on the force calculations.

Programming Language: C++

Known Portability Issues: The benchmark is written in conservative C++, is quite portable, and the inner loop code (module ComputeNonbondedUtil.C) contains no aliasing. The erfc() function is required for startup. On Windows, -DWIN32ERFC is defined during compilation to build a version of erfc() for little-endian, 32-bit and 64-bit platforms. This is only needed for startup, and should not affect overall performance.

References:

[1] www.ks.uiuc.edu/Research/namd/
[2] Laxmikant Kale, Robert Skeel, Milind Bhandarkar, Robert Brunner, Attila Gursoy, Neal Krawetz, James Phillips, Aritomo Shinozaki, Krishnan Varadarajan, and Klaus Schulten. NAMD2: Greater scalability for parallel molecular dynamics. *Journal of Computational Physics*, **151:283-312, 1999.**

447.dealII *(Floating point benchmarks cont'd)*

General Category: Solution of Partial Differential Equations using the Adaptive Finite Element Method

Authors: Wolfgang Bangerth, Guido Kanschat, Ralf Hartmann and other contributors cited at [1].

Description: The benchmark uses deal.II, a C++ program library targeted at adaptive finite elements and error estimation. The library uses state-of-the-art programming techniques of the C++ programming language, including the Boost library. It offers a modern interface to the complex data structures and algorithms required for adaptivity and enables use of a variety of finite elements in one, two, and three space dimensions, as well as time-dependent problems.

The main aim of deal.II is to enable development of modern finite element algorithms, using among other aspects sophisticated error estimators and adaptive meshes. The deal.II library provides the application programmer with grid handling and refinement, handling of degrees of freedom, input of meshes and output of results in graphics formats, and support for several space dimensions.

The testcase solves a Helmholtz-type equation with non-constant coefficients that is at the heart of solvers for a variety of applications. It uses modern adaptive methods based on duality weighted error estimates to generate optimal meshes. The equation is solved in 3d.

Input: Except for the number of refinement steps, the input is generated by code in the benchmark, using small modules that have been added to the library. This is a typical approach in many finite element applications, and is chosen for efficiency.

Output: The output describes the problem state at each refinement step. To reduce the amount of output, the benchmark downsamples the solution to a coarser grid In addition, the program prints the number of the iteration, the number of degrees of freedom in this iteration, the computed value for the point evaluation and an estimated error.

Programming Language: C++, incl. the Boost library [2\

Known Portability Issues: Because deal.II uses state-of-the-art C++ programming techniques, a number of minor errors have been found in various compilers. There is also one major portability issue regarding an unspecified item in the C++ standard: deal.II implements support for 1d, 2d, and 3d using a system of templates and explicit specializations. This problem is described in more detail at www.spec.org/cpu2006/Docs/447.dealII.html

References:

[1] http://www.dealii.org/
[2] http://www.boost.org/

450.soplex *(Floating point benchmarks cont'd)*

Authors: Roland Wunderling, Thorsten Koch, Tobias Achterberg

General Category: Simplex Linear Program (LP) Solver

Description: 450.soplex is based on SoPlex Version 1.2.1. SoPlex solves a linear program using the Simplex algorithm. The LP is given as a sparse m by n matrix A, together with a right hand side vector b of dimension m and an objective function coefficient vector c of dimension n. The matrix is sparse in practice. SoPlex employs algorithms for sparse linear algebra, in particular a sparse LU-Factorization and solving routines for the resulting triangular equation systems.

Input: test uses the "finnis" test problem from netlib [1], with 497 rows and 614 columns. Train uses rail582 [2] with 582 rows and 55,515 columns, and pds-20.mps [3] with 33,874 rows and 105,728 columns. Ref uses rail2586 with 2586 rows and 920,683 columns, and pds-50.mps with 83,060 rows and 270,095 columns. The test model is solved to full optimality; train and ref are solved until an iteration limit is reached.

Output: includes the objective function for the optimal solution or the value of the objective function after the iteration limit and the number of iterations performed.

Programming Language: ANSI C++

Known Portability Issues: none

References:

[1] http://www.netlib.org/lp/data
[2] J. E. Beasley's OR Library describes the rail problems http://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html and they are available at Mittelmann's rail directory http://plato.asu.edu/ftp/lptestset/rail/
[3] W. J. Carolan, J. E. Hill, J. L. Kennington, S. Niemi, S. J. Wichmann, "An Empirical Evaluation of the KORBX Algorithms for Military Airlift Applications" *Operations Research* vol. 38, no. 2 (1990), pp. 240-248. The pds-20 and pds-50 models were obtained from Hans Mittelman's website http://plato.asu.edu/bench.html .
[4] SoPlex: www.zib.de/Optimization/Software/Soplex
[5] Roland Wunderling, Paralleler und Objektorientierter Simplex-Algorithmus, ZIB report TR 96-09, Berlin 1996. www.zib.de/Publications/abstracts/TR-96-09
[6] Robert J. Venderbei, Linear Programming: Foundations and Extensions, Second Edition, Kluwer Academic Publishers, 2001.
[7] George Dantzig, Linear Programming and Extensions, Princeton University Press 1998, (1963).

453.povray *(Floating point benchmarks cont'd)*

Author: POV-Team, contact Thorsten Fröhlich

General Category: Computer Visualization

Description: POV-Ray is a ray-tracer. Ray-tracing is a rendering technique that calculates an image of a scene by simulating the way rays of light travel in the real world, but it does so backwards. In the real world, rays of light are emitted from a light source and illuminate objects. The light reflects off of the objects or passes through transparent objects. This reflected light hits the human eye or a camera lens. As the vast majority of rays never hit an observer, it would take forever to trace a scene. Thus, ray-tracers start with their simulated camera and trace rays backwards out into the scene. For every pixel rays are shot from the camera into the scene to see if it intersects with any of the objects in the scene. Every time an object is hit, the color of the surface at that point is calculated. For this purpose rays are sent to each light source to determine the amount of light coming from it or if the object is in shadow.

POV-Ray supports 30 different geometric objects including blobs, quadrics, spheres, cylinders, polygons, meshes, isosurfaces and constructive solid geometry. Intersections of rays with geometry objects are computed by solving complex mathematical equations directly or by numeric approximation algorithms. All these algorithms are sensitive to floating-point accuracy.

Input: 453.povray renders a 1280x1024 pixel anti-aliased image of a chessboard with all pieces in the starting position. The objects were selected to show various geometry objects available in POV-Ray. The input also generates a height field out of a simple fractal function and provides the function for an isosurface object. All objects in the scene have procedural textures that determine their surface texture. Many of these textures make use of a variant of the Perlin noise function. Some objects refract light and others are highly reflective.

Output: The output files are the rendered image and a log containing statistical information about the intersection tests performed and other related information.

Programming Language: ISO C++

Known Portability Issues: mathimf.h is included for the Intel compiler by using -DSPEC_CPU_WINDOWS_ICL.

References:

[1] POV-Ray (Persistence of Vision Raytracer) available at http://www.povray.org

[2] Andrew S. Glassner: An Introduction to Ray tracing. Academic Press 1989, ISBN 0-12-286160-4.

454.calculix *(Floating point benchmarks cont'd)*

Author: Guido D.C. Dhondt

General Category: Structural Mechanics

Description: 454.calculix is based on CalculiX, a free software finite element code for linear and nonlinear three-dimensional structural applications. It uses classical theory of finite elements described in books such as [4]. CalculiX can solve problems such as static problems (bridge and building design), buckling, dynamic applications (crash, earthquake resistance) and eigenmode analysis (resonance phenomena). .

Input: a mesh describing the geometry of the structure, material properties, geometric boundary conditions and natural boundary conditions. The reference example describes the deformation of a compressor disk due to centrifugal load. The compressor disk is rotated at high speed and deformation and stresses are analyzed. Only a segment describing one seventh of the disk was modelled. The other six seventh are taken into account by cyclic symmetry conditions. The material of the disk is a combination of linear elastic, viscoelastic, Ogden-type and Ramberg-Osgood-type material. Target of the calculation is the displacement and stress field at full speed.

Output: The output consists of variable fields across the structure, usually the displacements and the stresses.

Programming Language: 454.calculix uses Fortran 90 and C. SPOOLES [7], the mathematical code to solve the set of linear equations generated by CalculiX CrunchiX, in C.

Known Portability Issues: If your C compiler does not support the C99 format %zd, try compiling with -DSPEC_CPU_NOZMODIFIER

References:

[1] www.calculix.de

[2] www.dhondt.de

[3] Dhondt, G., "The Finite Element Method for Three-Dimensional Thermomechanical Applications", Wiley, 2004.

[4] Zienkiewicz, O.C. and Taylor, R.L., "The Finite Element Method", Fourth Edition, McGraw Hill, 1989

[5] Belytschko, T., Liu, W.K.L. and Moran, B., "Nonlinear Finite Elements for Continua and Structures", Wiley, 2000.

[6] Hughes, T.J.R., "The Finite Element Method", Dover, 2000.

[7] SPOOLES: http://netlib.bell-labs.com/netlib/linalg/spooles/spooles.2.2.html

459.GemsFDTD *(Floating point benchmarks cont'd)*

Authors: Ulf Andersson and others at the Parallel and Scientific Computing Institute (PSCI) in Sweden.

General Category: Computational Electromagnetics (CEM)

Description: solves the Maxwell equations in 3D in the time domain using the finite-difference time-domain (FDTD) method. The radar cross section (RCS) of a perfectly conducting (PEC) object is computed. GemsFDTD is a subset of GemsTD from GEMS (General ElectroMagnetic Solvers).

The core of the FDTD method are second-order accurate central-difference approximations of Faraday's and Ampere's laws. These central-differences are employed on a staggered Cartesian grid resulting in an explicit finite-difference method. An incident plane wave is generated using so-called Huygens' surfaces. This means that the computational domain is split into a total-field part and a scattered field part, where the scattered field part surrounds the total-field part. The computational domain is truncated by an absorbing layer in order to minimize the artificial reflections at the boundary. The Uni-axial perfectly matched layer (UPML) by Gedney is used here. A time-domain near-to-far-field transformation computes the RCS according to the Martin and Pettersson.

The train case uses a smaller computational domain and a thinner absorbing-boundary-condition layer.

Input: The inputs define problem size, number of time steps, cell size, CFL value, the excitation, an incident plane wave, the absorbing layer, and the near-to-far-field transform.

Output: The output is an ASCII file containing the requested RCS data Two included Matlab scripts can be used for plotting (though they are not used by SPEC). For the PEC sphere, an analytical reference solution is supplied.

Programming Language: Fortran 90

Known Portability Issues: None

References:

[1] www.psci.kth.se/Programs/GEMS/

[2] Allen Taflove, Computational Electrodynamics: The Finite-Difference Time-Domain Method, Artech House, 2000

[3] T. Martin and L. Pettersson, IEEE Trans. Ant. Prop. Vol. 48, No. 4, pp. 494-501, Apr. 2000.

[4] S. Gedney, IEEE Trans. Ant. Prop., vol. 44, no. 12, pp 1630-1639, Dec. 1996.

[5] A report on a subset of GemsFDTD may be found at http://www.pdc.kth.se/info/research/trita/PDC_TRITA_2002_1.pdf

465.tonto *(Floating point benchmarks cont'd)*

Authors: Daniel J. Grimwood and Dylan Jayatilaka

General Category: Quantum Crystallography

Description: Tonto is an open source quantum chemistry package. Objectives include simplicity, portability, and extensibility. Tonto is written in an object oriented design, in Fortran 95. It uses derived types and modules to represent classes. Classes range from integers and text files, to atoms, spacegroups and molecules. Tonto uses dynamic memory and array operations.

The profiles of Tonto calculations are typical of *ab initio* quantum chemistry packages: a large portion is dedicated to the evaluation of integrals between products of Gaussian basis functions. The SPEC reference calculation is in the field of quantum crystallography. It places a constraint on a molecular Hartree-Fock wavefunction calculation to better match experimental X-ray diffraction data. It is expected that other similar properties calculated from the constrained wavefunction should also agree better with experiment.

Input: The input file contains the crystal structure, atom positions, and basis functions, and experimental X-ray diffraction data. It then gives calculation parameters, and does the calculation. The crystal structure, atom positions, and X-ray data are from the literature.

Output: The main output file is regularly updated to show how far the calculation is from the final answer. Once the final model wavefunction is obtained, the X-ray diffraction data calculated from it are printed out together with the experimental data and compared. The chi^2 of one means the calculated and experimental data sets agree to within experimental accuracy.

Programming Language: Fortran 95

Known Portability Issues: Tonto makes extensive use of new features added with the Fortran 90/95 standards such as generic interfaces and vector subscripts. Some older compilers may not support these features.

References:

[1] Tonto home page - www.theochem.uwa.edu.au/tonto/

[2] D. Jayatilaka and D. J. Grimwood, Computational Science - ICCS 2003, 2660, 142-151, (2003)

[3] D. Jayatilaka and D. J. Grimwood, Acta Cryst., A57, 76-86, (2001)

[4] M. Messerschmidt, A. Wagner, M. W. Wong and P. Luger, J. Am. Chem. Soc., 124(5), 732-733, (2002)

[5] D. J. Grimwood, I. Bytheway and D. Jayatilaka, *J. Comp. Chem.*, **24(4)**, 470-483, (2003)

470.lbm *(Floating point benchmarks cont'd)*

Author: Thomas Pohl

General Category: Computational Fluid Dynamics, Lattice Boltzmann Method

Description: This program implements the so-called "Lattice Boltzmann Method" (LBM) to simulate incompressible fluids. It is the computationally important part of a larger code use in material science to simulate fluids with free surfaces, in particluar the formation and movement of gas bubbles in metal foams [1]. For benchmarking purposes, the code makes extensive use of macros which hide the details of the data access. A visualization of the submitted code can be seen at www.spec.org/cpu2006/Docs/470.lbm.html

Input: number of time steps and choice among two basic simulation setups, lid-driven cavity (shear flow driven by a "sliding wall" boundary condition) and channel flow (flow driven by inflow/outflow boundary conditions)

The basic steps of the simulation code are: If an obstacle file was specified it is read and the obstacle cells are set accordingly. The specified number of time steps are calculated in the selected simulation setup (lid-driven cavity or channel flow). Depending on the action chosen the result is either stored, compared to an existing result file, or thrown away. In the Lattice Boltzmann Method, a steady state solution is achieved by running a sufficient number of model time steps.

For the reference workload, 3000 time steps are computed. The test and training workloads, use a smaller number of steps. The geometry used in the training workload is different from the geometry used in the reference workload. Also, the reference workload uses a shear flow boundary condition, whereas the training workload does not. Nevertheless, the computational steps stressed by the training workload are the same as those stressed in the reference run.

Output: the 3D velocity vector for each cell.

Programming Language: ANSI C

Known Portability Issues: None

References:

[1] The FreeWiHR homepage http://www10.informatik.uni-erlangen.de/en/Research/Projects/FreeWiHR

[2] Y.-H. Qian, D. d'Humieres, and P. Lallemand: *Lattice BGK models for Navier-Stok*es equation. Europhys. Lett. 17(6): 479-484, 1992

[3] Thomas Pohl, Markus Kowarschik, Jens Wilke, Klaus Iglberger, and Ulrich Rüde: Optimization and Profiling of the Cache Performance of Parallel Lattice Boltzmann Codes. Parallel Processing Letter 13(4) 549-560, 2003, postscript copy available on the SPEC media

481.wrf *(Floating point benchmarks cont'd)*

Authors: National Center for Atmospheric Research (NCAR) in collaboration with multiple government agencies, universities, and many others [1].

General Category: Weather Forecasting

Description: 481.wrf is based on the Weather Research and Forecasting (WRF) Model, which is a next-generation mesocale numerical weather prediction system designed to serve both operational forecasting and atmospheric research needs.

WRF features multiple dynamical cores, a 3-dimensional variational (3DVAR) data assimilation system, and a software architecture allowing for computational parallelism and system extensibility. WRF is suitable for a broad spectrum of applications across scales ranging from meters to thousands of kilometers.

The parallel portions of the code have been turned off for SPEC CPU2006 as the interest here is in single processor performance. WRF version 2.0.2 is used in the benchmark version, 481.wrf.

Input: The WRF Standard Initialization (SI) software is used to create the data sets.

The June 2001 SI data archived at NCAR is used as data sets. The data is 10 km from 1200 UTC June 11 - 1200 UTC Jun 12 2001 at 3 h interval.

Output: The temperature at a certain grid point is printed for every time step and validated.

Programming Language: Fortran 90 and C

Known Portability Issues: 481.wrf requires the netCDF library for I/O. netCDF is packaged as part of the source, but may need to be configured for your system. SPEC_CPU portability options are provided.

481.wrf uses unformatted I/O to read its data files. By default, files with 4-byte headers will be read. If your system requires 8-byte headers, please set "wrf_data_header_size = 8" in the 481.wrf section of your config file.

References:

[1] The wrf development teams are listed at http://www.wrf-model.org/development/development.php. Wrf uses netcdf, which has many contributors as detailed at http://www.unidata.ucar.edu/software/netcdf/credits.html

[2] http://www.wrf-model.org/index.php

482.sphinx3 *(Floating point benchmarks cont'd)*

Authors: Sphinx Speech Group, Carnegie Mellon University [1]. Thank you especially to CMU Researchers Evandro Gouevea, Arthur Chan, and Richard Stern for their assistance to SPEC in creating this version. Thanks also to Paul Lamere of Sun for timely consulting on many porting questions.

General Category: Speech Recognition

Description: Sphinx-3 is a widely known speech recognition system from Carnegie Mellon University.

This description assumes that the reader has already seen the Sphinx 3 introduction [2], which provides an excellent introduction to the inputs, outputs, and operation of the code. (A copy of this file as of mid-2005 is included in the SPEC CPU2006 kit).

CMU supplies a program known as livepretend, which decodes utterances in batch mode, but otherwise operates as if it were decoding a live human. In particular, it starts from raw audio, not from an intermediate (cepstra) format. Although in real life IO efficiency is obviously important to any speech recognition system, for SPEC CPU purposes we wish to concentrate on the CPU-intensive portions of the task. Therefore, main_live_pretend.c has been adapted as spec_main_live_pretend.c, which reads all the inputs during initialization and then processes them repeatedly with different settings for the "beams" (the probabilities that are used to prune the set of active hypotheses at each recognition step).

Input: The AN4 Database from CMU [3] is used. The raw audio format files are used in either big endian or little endian form (depending on the current machine).

Output: Correct recognition is determined by examination of which utterances were recognized (see lines "FWDVIT" in the generated .log files), as well as a trace of language and acoustic scores.

Programming Language: C

Known Portability Issues: None

References:

[1] http://www.speech.cs.cmu.edu/
[2] http://cmusphinx.sourceforge.net/sphinx3/s3_description.html
[3] www.speech.cs.cmu.edu/databases/an4
[4] http://cmusphinx.sourceforge.net/sphinx3/

999.specrand *(floating point set),* and 998.specrand *(integer set)*

Author: Cloyce D. Spradling

General Category: Mine Canary

Description: specrand is a small harness for the algorithm presented in [1]. The datatypes used have been modified so that the algorithm will work properly on systems with 64-bit longs The benchmark simply generates a sequence of pseudorandom numbers starting with a known seed.

This benchmark is not a timed component of CPU2006; rather, it's there as an indicator of larger problems. Several of the other benchmarks use the specrand code as their PRNG. Thus, a failure in 999.specrand would point out a source of error in those codes as well. This is cheap (in both time and space) insurance.

Input: 999.specrand's input consists of two numbers: a seed value for the PRNG, and a count of the numbers to generate.

Output: The specified number of random numbers are generated and output twice. The first set is unscaled output from the PRNG output as a standard floating point number with no size or precision modifiers (printf '%f' format). The second set is scaled to between 1 and 2048000, and is output as a standard integer with no size modifiers (printf '%d' format). The PRNG is not re-seeded between sequences, so actually count*2 numbers are generated.

Programming Language: ANSI C

Known Portability Issues: This code probably will not work on a system where the standard 'int' type is 64-bits wide.

References:

[1] S. K. Park and K. W. Miller: "Random number generators: good ones are hard to find" October 1988. Communications of the ACM vol. 31 #10. http://doi.acm.org/10.1145/63039.63042