

# Monte Carlo Testdrive (MCT) package

This application demonstrates a simple Monte Carlo numerical integration with a simple non-programming interface and a massively parallel (ClearSpeed exploiting) implementation.

The computation takes the form of an expression to evaluate and a number of times to perform the evaluation so that an average, or mean, can be found.

This package demonstrates:

- An alternative programming model
- Integration with Microsoft's Excel
- Use of ClearSpeed's Vector Math and Random Number Generator libraries

This package is a simple combination of various technologies and software engineering to demonstrate a trivial user interface that will automatically exploit the underlying parallel machine. It does not provide a demonstration of achieving peak performance for the target application. Nor does it intend to be a new and robust numerical integration implementation.

## Techniques employed or Key-words

Dynamic compilation; Virtual machine; Microsoft Excel front-end; Vector Math Library; Random Number Generator; Stochastic methods; Monte Carlo; Integration.

### ***Dynamic Compilation***

Associated files: *middle/grammar.y middle/lexer.l middle/codegen.c middle/checktree.c middle/error.c middle/parse.c.*

The input, or problem specification is the simple composition of basic mathematical functions with random number generation facilities. This is processed in a typical manner to produce a program that is executable on the host and/or ClearSpeed processors. Given the narrow scope of the input language, and all code generation being run on the host, this is trivial in terms of run-time.

The input language has a simple form, like a single mathematical expression, with the following syntax and composition properties:

Italicised letters (*x,y,z ...*) denote expressions within a composite expression.

Constants (such as *3.14159*) are expressions.

*rand()* is an expression, it is a function with no arguments that produces a uniform random value in the range [0,..1).

*gauss(x)* is an expression, it is a function of one argument *x*, that must be an expression in the range [0,..1), that produces the Gaussian distributed form of *x*.

*x+y*, *x-y*, *x\*y* and *x/y* are expressions of two valid expression arguments *x* and *y* that produce the natural interpretation of the operator (addition, difference, and so on).

*(g)* is an expression of one argument *g* that permits the explicit ordering of evaluation in combined expressions such as *x+(y\*z)* and *(x+y)\*z*.

## Monte Carlo Testdrive (MCT) package

$\exp(x)$  and  $\ln(x)$  are expressions of one argument  $x$  that produce the values  $e^x$  and the natural log of  $x$  respectively.

$\sin(x)$  and  $\cos(x)$  are expressions of one argument that produce the sine and cosine of  $x$  respectively.

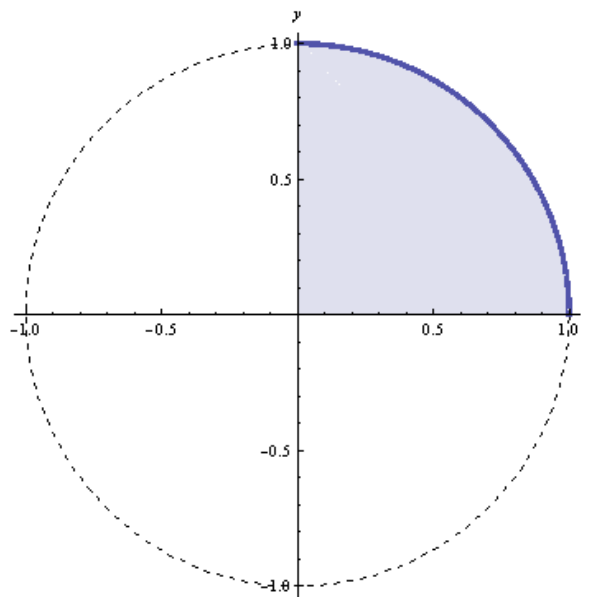
$\sqrt{x}$  is an expression of one argument that produces the square root of  $x$ .

$\max(x,y)$  and  $\min(x,y)$  are expressions of two arguments that select and produce the maximum and minimum of  $x$  and  $y$  respectively.

$\text{iflt}(x,y,z)$  is an expression of three arguments that selects and produces  $y$  if  $x < 0$  otherwise  $z$ .

### Example problems

We may write the integral of the function  $x^2 + y^2 = 1$  for the  $x$  and  $y$  range of  $[0, \dots, 1]$ . The known result is  $\pi/4$  as it is one quarter of the unit circle as illustrated below.



The blue line is where  $x^2 + y^2 = 1$  and the filled area is its integral. The unit circle is dashed and has a whole area equal to  $\pi$ , making the filled area  $\pi/4$ .

We can use the expression  $\text{iflt}(\exp(2*\log(\text{drand48}())) + \exp(2*\log(\text{drand48}())) - 1, 1, 0)$  that will result in the value 1 on those occasions where a randomly selected pair of  $x$  and  $y$  coordinates (in the range  $[0, \dots, 1]$ ) fall below the line, and 0 otherwise. We know that the probability of the randomly selected point will fall below the line is 1 in  $\pi/4$ . Choosing a large number of trials, accumulating the sum of each expression evaluation, and then dividing that by the total number of trials should tend towards the true value of  $\pi/4$ .

With a numerical value of 0.785398 for  $\pi/4$ , the following table shows how increasing the number of trials corresponds to an increasing convergence to the true value.

## Monte Carlo Testdrive (MCT) package

Number of trials	Accumulated sum/number of trials
9216	0.78385417
18432	0.79427083
36864	0.78428819
73728	0.77886285
147456	0.78005642
294912	0.78499349
589824	0.7846951
1179648	0.78618707
2359296	0.7859904
4718592	0.78560045
9437184	0.78506639
18874368	0.78570133
37748736	0.78555552
75497472	0.78560511
150994944	0.78544701

The properties of convergence are a function of the random number generator selected. This example uses *drand48()* in the underlying implementation, a pseudo-random number generator in the ClearSpeed Vector Math Library. The library offers other number generator implementations.

The expression used combined a pair of *exp* and *ln* to construct the  $x^2 + y^2$  parts. We could equally exchange this for *iflt(drand48()-sqrt(1-exp(2\*log(drand48())))-1,1,0)* by rewriting  $x^2 + y^2 = 1$  as  $y = \sqrt{1-x^2}$ .

The above example allows the validation or calibration of the system. Then, when integrals with no analytic or closed form are written (such as the integral of  $\sin^2 x$  over the range  $[0, \dots, 1)$ ) we can attach a degree of confidence to the result generated by this implementation.

### Virtual machine

Associated files: *backend/run\_host\_mc.c backend/run\_mc.c backend/mckernel.cn*

A simple byte-code is generated by the compiler. This is interpreted by a virtual machine running on the ClearSpeed boards. The program, along with the number of trials to be run is spread over the ClearSpeed boards, executed, and results collected to return to the user.

There are host and board versions of the virtual machine to manage any left-over trials that do not evenly spread over the ClearSpeed boards. The virtual machine instruction set is very similar to the compiler language. In a more advanced example, the compiler language would be considerably richer.

The virtual machine is stack-based and has the following instruction set (from *include/MCTISET.h*):

```
/* T = top of stack index S = stack */
/* P = program PC = program counter */
/* cpool = pool of constants */
```

## Monte Carlo Testdrive (MCT) package

```
/* Implicit is PC++, MCTINST_CONST does extra PC++ below as index
   to constant pool is stored in the program */
/* Stack pops are included in the mutli-operand instruction semantics
   */
MCTINST_ADD      /* T--; S[T]+=S[T+1] */
MCTINST_SUB      /* T--; S[T]-=S[T+1] */
MCTINST_DIV      /* T--; S[T]=S[T]/S[T+1] */
MCTINST_MUL      /* T--; S[T]=S[T]*S[T+1] */
MCTINST_SIN      /* S[T]=sin(S[T]) */
MCTINST_COS      /* S[T]=cos(S[T]) */
MCTINST_LOG      /* S[T]=log(S[T]) */
MCTINST_SQRT     /* S[T]=sqrt(S[T]) */
MCTINST_URAND    /* S[T]=drand48() */
MCTINST_GAUSS    /* S[T]=gauss(S[T]) */
MCTINST_MAX      /* T--; S[T]=max(S[T],S[T+1]) */
MCTINST_MIN      /* T--; S[T]=min(S[T],S[T+1]) */
MCTINST_CONST    /* S[T]=cpool[P[PC+1]] ; PC++ */
MCTINST_EXP      /* S[T]=exp(S[T]) */
MCTINST_PUSH     /* T++ */
MCTINST_IFLT     /* T-=2; S[T] = (S[T]<0.0)?S[T+1]:S[T+2]; */
```

Efficient exploitation of the underlying machine resources (including ClearSpeed cards) is possible with high-level virtual machine instructions like `MCTINST_IFLT`. This avoids the unnecessary inefficiencies introduced when adopting an instruction set more similar to a RISC microprocessor. The traditional priorities used in designing instruction sets do not correspond to priorities for virtual machines.

This virtual machine is designed for demonstration purposes. The function is limited to executing encoded mathematical expressions repeatedly. Alternative implementations might consider a target language of the compiler to be native machine code, whether directly, or via an intermediate code like C. This is beyond the scope for this example package.

As highlighted above, a large number of simulations are required to achieve high precision results. This means the virtual machine implementation is able to exploit the vector math and random number generator libraries that make more efficient use of the ClearSpeed processors.

### **Excel Front-end**

Associated files: *frontend/Excel/ExcelToolboxWithIntegration.xls*

An Excel front-end is provided, showing the simple integration of user-side invocation and driving of the Monte-Carlo back-end. The program string is sent through a socket to a server with a number of ClearSpeed cards. This implementation permits a number of Excel workstations to share a ClearSpeed accelerated server.

The following excerpt, similar to that in *frontend/Excel/ExcelToolboxWithIntegration.xls* shows the Visual Basic code to drive the system:

```
sum = 0#
sumsq = 0#

expression = "iflt(exp(2*log(drnd48()))+exp(2*log(drnd48()))-
1,1,0)"

error = MCT(expression, iterations, sum, sumsq)
```

# Monte Carlo Testdrive (MCT) package

`mean = sum / (Double) iterations`

The function `MCT` takes as input the expression and the number of trials, returning the accumulated sum (`sum`) and the accumulated square of each sum (`sumsq`). The `sumsq` component permits the calculation of an error or confidence interval.

## Package manifest

File	Short description
<i>backend/mckernel.cn</i>	CSX600 Cn code for virtual machine.
<i>backend/run_host mc.c</i>	Host C code for virtual machine.
<i>backend/run_mc.c</i>	Host code to manage CSX600 processors running the virtual machine(s).
<i>middle/checktree.c</i>	Compiler helper routines.
<i>middle/codegen.c</i>	Byte-code generation helper routines.
<i>middle/error.c</i>	Compiler helper routines.
<i>middle/grammar.y</i>	Compiler language specification.
<i>middle/lexer.c</i>	Compiler helper (lexical analyser specification).
<i>middle/parse.c</i>	Compiler driver routines.
<i>frontend/Excel/main.c</i>	Server implementation that accepts expressions over sockets from the Excel spreadsheet clients and drives the compiler & virtual machines.
<i>frontend/Excel/ExcelToolboxWithIntegration.xls</i>	Excel spreadsheet with calls to the Visual Basic macros to demonstrate their use.
<b>Within the Excel spreadsheet under Tools→Macro→Visual Basic Editor</b>	
<i>CProtocol.cls</i>	Visual Basic socket helper routines.
<i>CProtocols.cls</i>	Visual Basic socket helper routines.
<i>SocketModule.bas</i>	Visual Basic socket helper routines.
<i>MCTModule.bas</i>	Visual Basic routines to manage the Excel spreadsheet requirements and communicate with the server that executes them.
<i>makefile.stand.alone</i>	Makefile for Linux based machines.

## Building and running MCT

A Linux compatible makefile is provided and has been tested with ClearSpeed's SDK version 2.51. The following two commands should suffice to build MCT:

```
$> source /opt/clearspeed/csx600_m512_le/bin/bashrc
$> make -f makefile.standalone
```

MCT is run as a service on a ClearSpeed accelerated host, and any number of clients viewing the Excel spreadsheet can share the resource. To launch the server, on the Linux host:

```
$> ./MCTSocketServer 12345
```

## Monte Carlo Testdrive (MCT) package

Where *12345* is a socket number chosen for this service. This should correspond to the socket sought in the Visual Basic module within the Excel spreadsheet (see MCTModule.bas, in the function MCT). The server should start with a message similar to:

```
Found 8 cards.  
Found 2 processors on card 8.  
Found 1536 pes in total  
Started socketserver. Awaiting connections...
```

The output will vary depending on the number of ClearSpeed X620 or e620 accelerators you have installed. This application has been tested with at least eight cards in one system.

The client spreadsheets require configuration also, and the MCT function in the MCTModule.bas file contains two parameters. MCTModule.bas is found by selecting *Tools*→*Macro*→*Visual Basic Editor* from the menu within Excel. The following line requires adjustment to your system configuration.

```
lngRetVal = vbConnect(lngSocket, "192.168.40.150", 12345)
```

The IP address should be set to the name (or IP address) of the server running MCTSocketServer, and the socket number should correlate with that used to launch the server.

When the spreadsheet is open, you may alter parameters in the cells such as the number of simulations, and the sheet will invoke the server to recalculate with the new number of simulations. There are integration examples and Monte Carlo expressions for European call options included in the spreadsheet for illustration.