LA-UR-14-27037
Approved for public release; distribution is unlimited.

Title:	New Hash-based Energy Lookup Algorithm for Monte Carlo Codes
Author(s):	Brown, Forrest B.
Intended for:	OECD-NEA-WPNCS Expert Group Meeting - Advanced Monte Carlo Techniques, 2014-09-15/2014-09-19 (Paris, France) MCNP documentation
Issued:	2014-09-08

New Hash-based Energy Lookup Algorithm for Monte Carlo Codes

Forrest Brown

Monte Carlo Codes Group, XCP-3
X Computational Physics Division
Los Alamos National Laboratory

Abstract

New Hash-based Energy Lookup Algorithm for Monte Carlo Codes

Forrest Brown, XCP-3, LANL

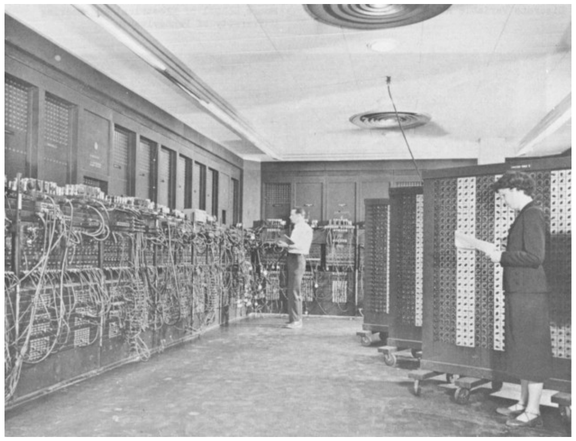
This talk provides a new algorithm for energy lookups during the construction of material cross-sections in a continuous-energy Monte Carlo code. A new hash-based energy lookup algorithm provides speedups of 15-20x over conventional schemes and requires about 1,000x less memory than unified grid methods. The hashing scheme is based on a log-energy grid and provides search bounds for each isotope that greatly reduce the lengths of energy table searches. It should be useful to code developers for optimizing the performance of any Monte Carlo code for particle transport.

Outline

- **Introduction**
- **Table Searches**
  - Total cross-section – {search, interpolate, accumulate}
  - Unified energy grid schemes
- **Hash-based Energy Lookup Algorithm**
  - History
  - Setup
  - Usage
- **Test Results**
  - Stand-alone
  - MCNP6.1.1
- **Conclusions**

Introduction

Monte Carlo & MCNP History



ENIAC – 1945

30 tons

20 ft x 40 ft room

18,000 vacuum tubes

0.1 MHz

20 word memory

patchcords

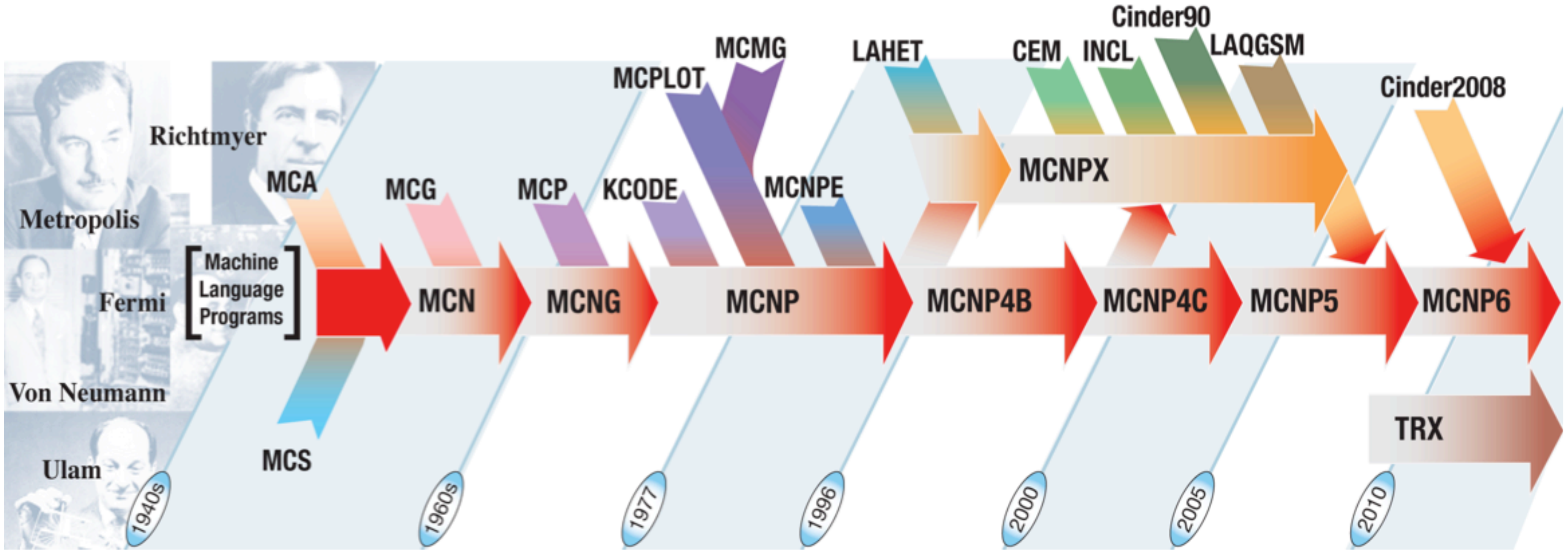
Manhattan Project – 1945...

Discussions on using ENIAC

Ulam suggested using the "method of statistical trials"

Metropolis suggested the name "Monte Carlo"

Von Neumann developed the first computer code



MCNP6 Features

mcnp5
neutrons, photons, electrons
cross-section library physics
criticality features
shielding, dose
"low energy" physics
V&V history
documentation

New Criticality Features
Sensitivity/Uncertainty Analysis
Fission Matrix
OTF Doppler Broadening

Fission
MCNP5/X multiplicity
LLNL fission package
CGM/LLNLGAM, CGMF (soon)

mcnp6

mcnp6
protons, proton radiography
high energy physics models
magnetic fields

Partisn mesh geometry
Abaqus unstructured mesh

mcnpx
33 other particle types
heavy ions
CINDER depletion/burnup
delayed particles

High energy physics models
CEM, LAQGSM, LAHET
MARS, HETC

Continuous Testing System
~10,000 test problems / day

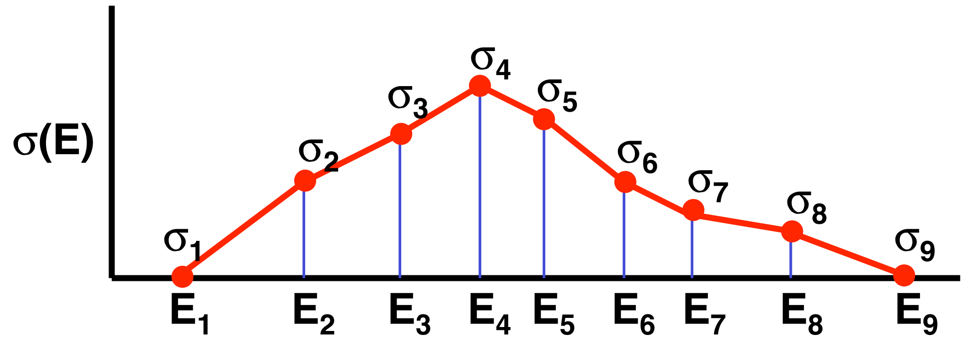
mcnp5 – 100 K lines of code
mcnp6 – 500 K lines of code

mcnp6.1 – 2013
mcnp6.1.1b – 2014

Table Searches

Piecewise Linear Data

- **Cross-section data are stored as piecewise linear functions of E**
  - **Typical $\sigma(E)$ vs E**



- **Usually stored as linear arrays:**

  N = number of entries

  E(1..N) = array of E values = ($E_1, E_2, \ldots, E_N$)

  σ(1..N) = array of σ values = ($\sigma_1, \sigma_2, \ldots, \sigma_N$)

- **Two steps are required to lookup & use the data:**
  1. **Given E, search the E() array to find interval k containing E ($1 \le k \le N-1$)**
  2. **Interpolate linearly between E_k & E_{k+1}**

$$\sigma(E) = \sigma_k + \left(\frac{E - E_k}{E_{k+1} - E_k} \right) \cdot \left(\sigma_{k+1} - \sigma_k \right), \; E_k \le E \le E_{k+1}$$

Example

- After a collision (before a flight) or entering new material
  - Must look up & interpolate σ_T for the neutron energy E, for each nuclide in a material
  - The σ_T's are used to determine Σ_T for the material
  - Σ_T is then used in randomly sampling of distance to collision

```
For    U235, U238, O16, …    (fuel material)
.
.            Search the array of energies for the nuclide, find interval k containing E
.            Interpolate $\sigma_T$ for nuclide at energy E
. . .
Add   N$\sigma_T$'s   for all nuclides in material to get  $\Sigma_T$
```

  - Similar interpolate & accumulate for scattering, absorption, fission, …..

- This set of operations {search, interpolate, accumulate} often consumes 1/3 – 2/3 of the overall time in neutron transport MC

Search Algorithms

- There is extensive literature on search algorithms
  - D.E. Knuth, The Art of Computer Programming Vol 3 - Sorting & Searching
  - Many other references - books & journals
- For general Monte Carlo codes, the commonly-used methods are linear search &/or binary search of the cross-section energy tables
  - Need 1 table search for each of the nuclides in a material
  - Linear search takes O(N) time, best when N ~ 10 or less
  - Binary search takes O(*ln* N) time, best when N ~ large
- To reduce the time needed for the table searches for cross-section data, several unified energy grid schemes were used in the past
  - Map the data for every nuclide in the problem onto 1 energy grid
  - Requires only 1 energy table search, rather than 1 table search for every nuclide in a material
  - Can be 10-100x faster for energy lookups

Unified Energy Grid Schemes

- **Scheme 1 – very old** (racer, rcp, o5r, …)
  - Used in the 1960s – 1980s due to memory limitations
  - Typically 10^4 – 10^5 energy bins (supergrouped)
  - Map all xsec data to these bins
  - Approximate, required weighting functions
- **Scheme 2 – unified grid** (psg, serpent, …)
  - Combine all xsec energy grids, including all energy points
  - Expand all xsec data onto unified grid
  - Exact, but required very large amounts of memory
- **Scheme 3 – unified grid with pointers** (serpent, …)
  - Combine all xsec energy grids, including all energy points
  - For each unified grid bin, store pointers to bins in each nuclide xsec data set
  - Exact, retains original nuclide xsec data
  - Extra storage for unified grid & nuclide pointers
  - Requires only 1 table search, then (indirect) lookups in nuclide tables
- **Scheme 4 – NEW, current hash-based energy lookup** (mcnp611)

Unified Energy Grid Schemes – Memory Storage

	nuclides	E pts	ACE xs	Ugrid+xs	Ugrid+ptrs	NEW
K Smith bench	64	.73 M	.12 GB	1.5 GB	.38 GB	2.2 MB
Rx pin, 1 temp	77	.66 M	.12 GB	1.6 GB	.41 GB	2.6 MB
Rx pin, 2 temps	145	1.2 M	.24 GB	5.6 GB	1.4 GB	4.8 MB
Rx pin, 5 temps	349	2.8 M	.55 GB	31 GB	7.8 GB	12 MB
All nucs, 1 temp	423	2.6 M	.58 GB	36 GB	9.0 GB	14 MB

ACE xs = actual memory for ACE data in mcnp611
E pts = total energy points, summed over all ACE nucs = pts in Ugrid

Ugrid+xs = extra storage for unified E-grid + $\{\sigma_T, \sigma_A, \sigma_E$, heating $\}$ at each E & nuc
Ugrid+ptrs = extra storage for unified E-grid + pointers to nuc xsecs at each E & nuc
NEW = extra storage for current hash-based lookup, with 8192 ubins

Hash-based Energy Lookup Algorithm

New Hash-based Energy Lookup Algorithm

- **History**
  - Suggested by George Zimmerman (LLNL, ret.) in 2013
  - Used at KAPL for lattice physics code (Dave Austin) in ~1989, and in several variations in racer MC in 1980s
  - Certainly much older .....
- **Recent**
  - Zimmerman, in proprietary code mods, 2013
  - Brown, stand-alone & in mcnp6.1.1b, 2013-2014
- **Basic idea**

  Retain all mcnp6 machinery for energy lookups & forming the total cross-section, but

  ➜ use a physics-based hash scheme to greatly narrow the bounds for each binary search of nuclide E tables

  ➜ Minimal mcnp6 code changes, but significant speedups

  ➜ Modest memory storage, much less than unified grids

Setup

- **The setup portion of the algorithm, performed prior to neutron random walks, involves the following steps:**

1. **Determine E_{min} and E_{max} energy bounds for the problem**
   - **Check E_{max} & E_{min} for all nuclide ACE datasets in the problem**

2. **Setup the "ugrid" for the hashing function**
   - ***Ugrid*: uniform spacing in *ln(E)* between E_{min} and E_{max}**
   - ***M*: number of bins in *ugrid()*.**
   - **No need to store *ugrid*() -- just store M, E_{min}, E_{max}**
   - **mcnp611: $M = 8192$, reasonable speed/storage tradeoff**

3. **Setup nuclide search bounds for each ubin index**
   - **For each bin in *ugrid*, lookup & store for each nuclide the bounding indexes $k_1(u,n)$ and $k_2(u,n)$ in the ACE energy table for that nuclide (*n*= nuclide index, *N*= no. nuclides, *u*= index in *ugrid*)**
   - **Only need store $k_1(u,n)$, since $k_2(u,n) = k_1(u+1,n)+1$**
   - **Total extra storage $= (M+1) \cdot N \cdot 4$ bytes (int4 sufficient for ACE data)**

Note: The above steps do NOT involve any approximations

Usage During Random Walk Simulation

- **After particle energy change or when entering new material**

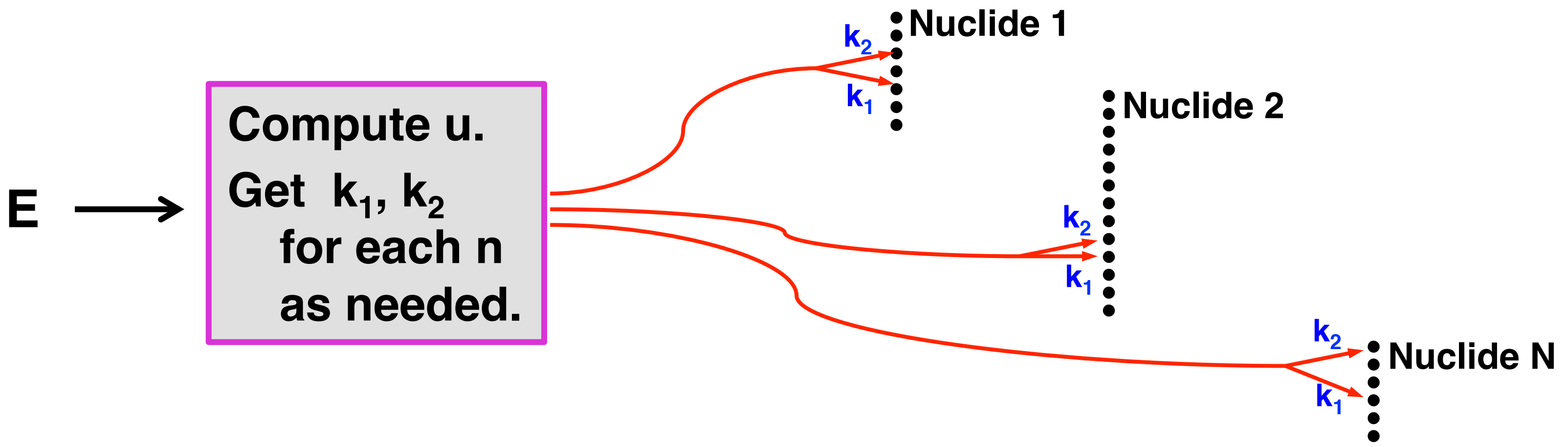
Defining $u_{min}= \log E_{min}$, $u_{max}= \log E_{max}$, $du = M / (u_{max} - u_{min})$

New algorithm for energy lookups for neutron energy *E* is:

$u = 1 + \lfloor du \cdot (\log E - u_{min}) \rfloor$, $\lfloor\ \rfloor$ **is truncation to the next lowest integer**

For each nuclide *n*:

search its energy table between entries $k_1(u,n)$ & $k_2 = k_1(u+1,n)+1$



Comments

- **Memory storage**
  - *ugrid* is completely defined by M, E_{min} , E_{max} -- need not be stored
  - Because $k_2(u,n) = k_1(u+1,n)+1$, the $k_2(u,n)$ values need not be stored
  - Total additional memory storage = $(M+1) \cdot N \cdot 4$ bytes
  - More compact memory use, so more cache-friendly
- **Speed/space tradeoff**
  - Larger M gives improved speed, but dependence is weak for $M > 1000$
  - Smaller M reduces speedup but also reduces memory requirements.
- **Choice of M does not in any way affect accuracy of the xsec data**
- **k_1 and k_2 indexes for each nuclide for each of the *ugrid* bins**
  - Bounds for performing ordinary binary searches in the nuclide ACE
  - These bounds narrow the range of the binary searches, so that only a small portion of each nuclide energy table need be searched
  - Frequently the search range in the nuclide energy tables is < 8. For such small ranges, a simple linear search will be slightly faster than a binary search & may provide additional small speedups

Timing Results

Stand-alone Testing (1)

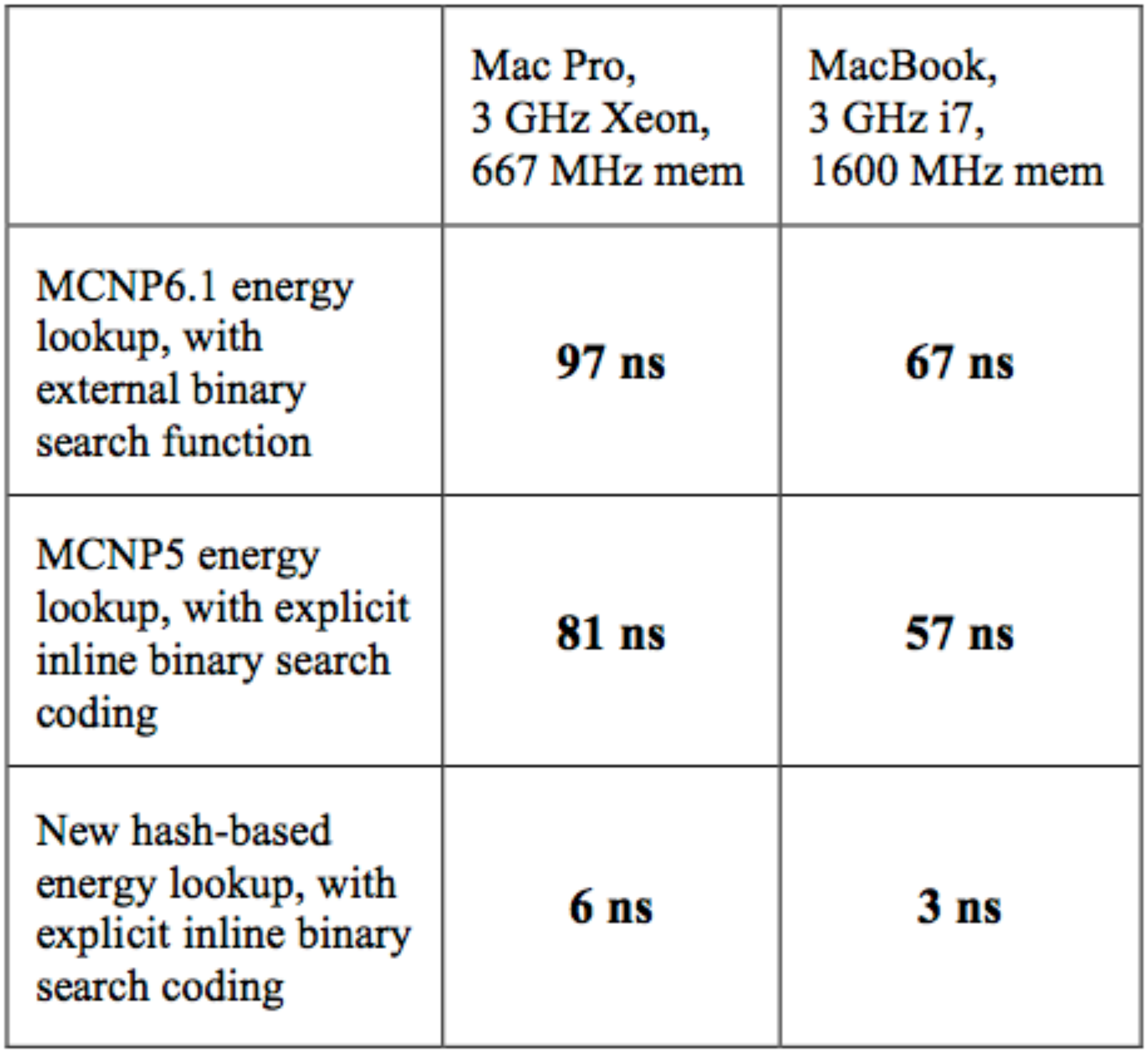
- **Stand-alone coding to compare 3 methods:**
  1. **Standard MCNP6.1 with external binary search function**
  2. **Standard MCNP5 with explicit inline coding for binary searches**
  3. **New hash-based scheme with inline binary searches.**

- **ACE datasets**
  - **The energy tables for 9 nuclides from the ENDF/B-VII.1 nuclear data libraries were used in the comparisons:**
    1001.80c, 8016.80c, 26056.80c, 92235.80c,
    92238.80c, 94239.80c, 94240.80c, 94241.80c, 6000.80c.
  - **These nuclides had energy table sizes ranging from 590 to 157,744 bins.**

- **For each energy lookup scheme, many millions of neutron energies were randomly sampled in the *ugrid* range, and then the energy lookups were performed for all 9 nuclides. Overall timing results are averages for the set of nuclides.**

Stand-alone Testing (2)

- **Timing results for stand-alone test of energy lookup methods. Results are the average time for each energy lookup**



	Mac Pro, 3 GHz Xeon, 667 MHz mem	MacBook, 3 GHz i7, 1600 MHz mem
MCNP6.1 energy lookup, with external binary search function	**97 ns**	**67 ns**
MCNP5 energy lookup, with explicit inline binary search coding	**81 ns**	**57 ns**
New hash-based energy lookup, with explicit inline binary search coding	**6 ns**	**3 ns**

- **Inlining binary searches gives 10-20% speedup (mcnp5 vs mcnp6.1)**
- **New hash-based scheme gives 15-20x speedup**
- **M = 8192 used for table**
- **Lookup time for other M on MacBook**

M = 64 k	**2 ns**
M = 32 k	**2 ns**
M = 16 k	**2 ns**
M = 8 k	**3 ns**
M = 4 k	**3 ns**
M = 1 k	**5 ns**

- **Mixed binary/linear search (break at 8) did not improve speedup**

MCNP6 Timing Tests (1)

- **MCNP6.1 (2013) runs significantly slower than MCNP5**
  - **Slowdowns are problem-dependent, 20% to 5x slower**

- **MCNP6.1.1 (2014)**
  - **Significant classic optimizations performed**

    Inline functions, eliminate non-unit-stride vector ops, if-guards, …
  - **New hash-based energy lookup scheme**
  - **Measured timing results for new energy lookup scheme compare mcnp6.1.1 before & after new scheme, with all other optimizations the same**

- **New energy lookup scheme provides 1 – 1.9x speedup in overall MCNP6.1.1 problem runtime** **(at least for neutron problems)**

- **MCNP6.1.1 is a lot faster than MCNP6.1**
- **MCNP6.1.1 is a little faster than MCNP5**

MCNP6 Timing Tests (2)

- **MCNP6.1.1 speedups due to new hash-based energy lookup algorithm**

Problem	Overall Code Speedup
OECD Performance Benchmark, 3D PWR, 60 isotopes, no tallies	**1.2x**
BAW XI(2), ICSBEP Problem LEU-COMP-THERM-008, Case-2, 31 isotopes, no tallies	**1.2x**
Godiva problem, 2 isotopes, no tallies	**1.0x**
Godiva problem, with trace amounts of 421 other isotopes	**1.9x**
Reactor pin cell with 147 isotopes	**1.5x**
Porosity tool for well-logging, 5 isotopes	**1.0x**

- **Speedup compares mcnp6.1.1 before & after new energy lookup scheme, with no other changes**
- **M = 8192 used for table**
- **All runs performed on Mac Pro (3 GHz, 2 quad-core) with 8 mcnp6 threads, using standard ENDF/B-VII data**

Summary

&

Conclusions

Summary & Conclusions

- **While the new hash-based energy lookup algorithm by itself is 15-20x faster than the conventional scheme, it is only a portion of the overall work and computations performed by MCNP.**
  - Overall speedups due to the new energy lookup algorithm will vary, depending on the particular physics and geometry of each problem.
  - Problems with more collisions per history & more isotopes per material will show larger speedups
  - Problems with more tallies, more complicated geometry, or fewer isotopes per material will show smaller or no speedups.

- **A particular advantage of the new lookup algorithm over unified energy grid schemes is the very significant reduction in memory requirements**
  - MBs instead of GBs
  - More cache-friendly
  - Significant memory savings are important for future generations of computers that are expected to have very many more processors, but less memory per processor.

- **The new lookup algorithm has been incorporated into MCNP6.1.1 (2014)**
  - Currently used only for energy lookups in neutron data tables.
  - Future work will investigate whether the new algorithm can be effective for other problems (e.g., photons, light ions).

Acknowledgments & References

Acknowledments

- Discussions of Monte Carlo algorithms and coding with George Zimmerman were, as always, stimulating and encouraging, and contributed significantly to the success of this work.
- This work was supported by the US DOE/NNSA Nuclear Criticality Safety Program and the Advanced Simulation & Computing Program.

References

- F.B. Brown, B.C. Kiedrowski, J.S. Bull, "Verification of MCNP6.1 and MCNP6.1.1 for Criticality Safety Applications," Los Alamos National Laboratory report, LA-UR-14-22480 (2014).
- Forrest Brown, Brian Kiedrowski, Jeffrey Bull, "MCNP5-1.60 Release Notes", Los Alamos National Laboratory report, LA-UR-10-06235 (2010).
- J.T. Goorley, et al., "Initial MCNP6 Release Overview - MCNP6 version 1.0," Los Alamos National Laboratory report, LA-UR-13-22934 (2013).
- J.R. Tramm & A.R. Siegel, “Memory Bottlenecks and Memory Contention in Multi-Core Monte Carlo Transport Codes,” Proceedings of SNA+MC 2013, Paris, France Oct 27-31 (2013).
- F.B. Brown, "Present Status of Vectorized Monte Carlo," *Trans. Am. Nucl. Soc*. 55, 323 (1987).
- J. Leppänen, "Two practical methods for unionized energy grid construction in continuous-energy Monte Carlo neutron transport calculation," *Ann. Nucl. Energy*, 36 (2009).
- J. Leppänen, A. Isotalo, "Burnup calculation methodology in the Serpent 2 Monte Carlo code," Proceedings of PHYSOR-2012, Knoxville, TN, Apr. 15-20 (2012).
- G. Zimmerman, private communication to F.B. Brown (2013).
- D. Austin, KAPL, private communication to F.B. Brown (~1989).

Miscellaneous

-

Interesting Information

Monte Carlo - Computer Operations & Performance

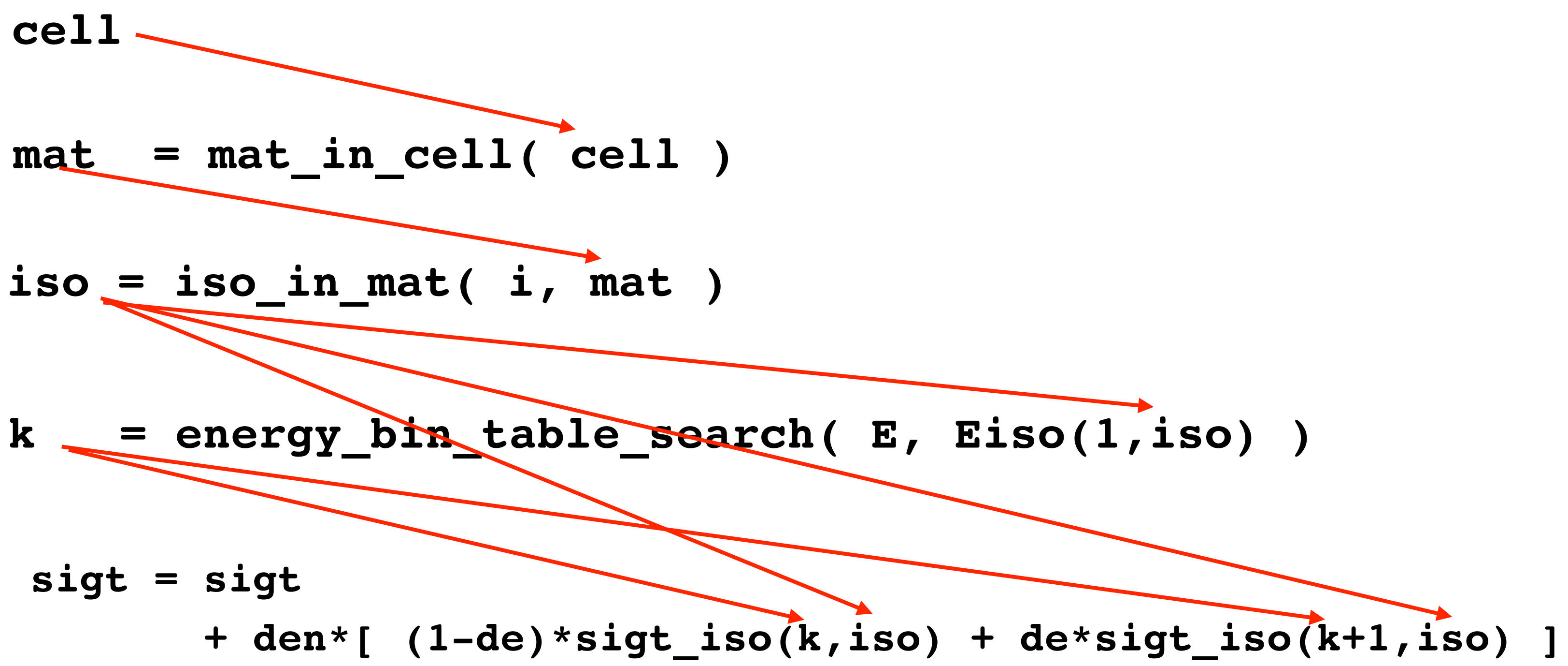
- **Breakdown of computer operations for typical large, general-purpose Monte Carlo code (approximate)**

40% -	**indexing, integer ops, memory access**
30% -	**test-and-branch**
25% -	**arithmetic**
5% -	**RN generation & sampling, 64-bit integers**

- **MC code performance vs. computer hardware**
  - **Memory access is largely random**
    - Little cache-coherency - only small gain from larger cache
    - Bus speed is important
  - **CPU-intensive, but not floating-point**
    - Big gains from multiple integer/logical functional units
    - Smaller gains from multiple floating-point units
  - **Compiler optimizations are critical**
    - Test-and-branch operations
    - Indexing & memory prefetching

Monte Carlo - Levels of Indirection

- **M.C. codes have many levels of indirection for memory access**



```
cell

mat   = mat_in_cell( cell )

iso = iso_in_mat( i, mat )


k    = energy_bin_table_search( E, Eiso(1,iso) )


 sigt = sigt
        + den*[ (1-de)*sigt_iso(k,iso) + de*sigt_iso(k+1,iso) ]
```

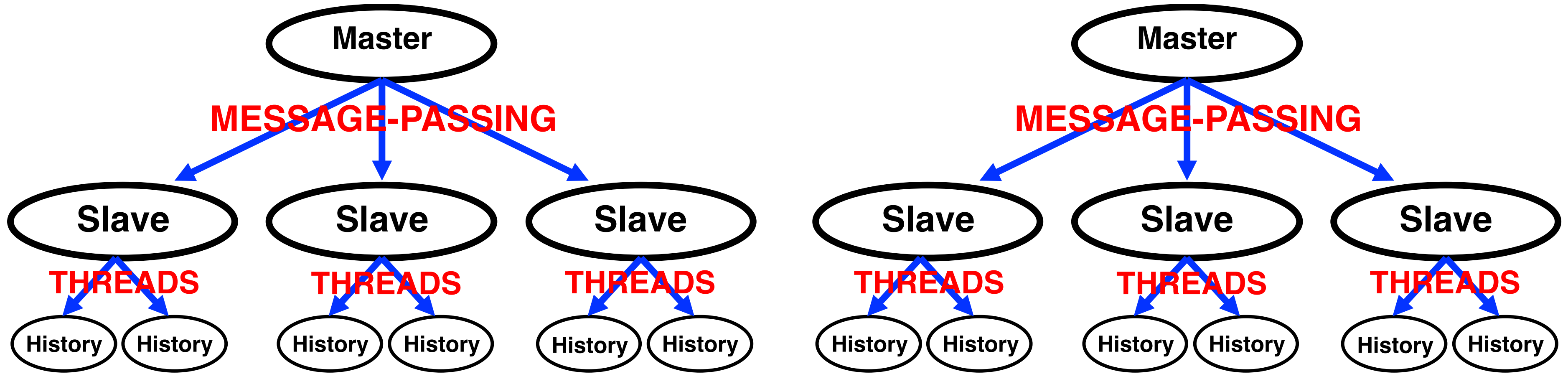
- **Each level of indirection:**
  - **Integer ops for indexing**
  - **Irregular memory access**
  - **Cache-misses**
  - **Inhibits pre-fetching, compiler optimization, & vectorization**

Computing - Latency & Threading

- **Hardware - Moore's Law**
  - Before 2000: 2X cpu speed every 18 months
  - After 2000: more cpu-cores per chip, not faster cpus
  - Today, hardware speed gains come from parallelism
- **Fast, multicore cpus**
  - Need more data & need it faster
  - Data transfer speed from memory to CPU has not kept up
  - Today, data access & latency are biggest concerns
- **Dealing with latency:**
  - Hardware -- cache, out-of-order execution, multicore, GPUs
  - Algorithms -- High-level, data order & layout, vectorization, threading
  - Important to match algorithms & hardware
- **Most large computer systems today are clusters**
  - Many nodes: fiber network interconnect
  - Multicore cpus: share memory within each node
  - Hierarchical parallelism for Monte Carlo

MCNP - Hierarchical Parallelism

Concurrent Jobs ➜



Parallel Processes

– Total processes = (# jobs) x (# MPI processes) x (# threads)

– Tradeoffs:

More MPI processes -	lots more memory & messages
More threads -	contention from lock/unlock shared memory
More jobs -	system complexity, combining results