Fast Generation of High-Quality Pseudorandom Numbers and Permutations Using MPI and OpenMP on the Cray XD1

Stephen Bique
Computer Scientist
Center for Computational Sciences

Naval Research Laboratory
Washington, D.C.
Outline

- Introduce MCG
- Choose constants for an MCG “in parallel”
- Choosing the modulus
- Performance benefit
- Choosing multiplicative constant
  - How to find candidates
  - How to test them
- Examples
- Calculate $\pi$ in parallel
- Summary
Multiplicative Congruential Generator (MCG)

\[ x_{n+1} = \alpha \cdot x_n \mod m \]
Choose constants:

- Generate pseudorandom numbers in parallel
  - Use different seeds, or
  - Use different multipliers
- Fast implementation
- Satisfactory statistical results
- Long period

MCG: \[ x_{n+1} = \alpha \cdot x_n \mod m \]
Choose Modulus

- **Prime**
  - Theoretical basis for choosing multiplicative constant $\alpha$
  - Our preliminary results show some moduli are better choices than others

- Sufficiently close to a power of two
  - Faster implementation than library call
  - Slightly faster for Mersenne primes
Algorithm 1
MCG Computation for Mersenne Primes

1. $X \leftarrow \alpha \cdot X$

2. $X \leftarrow \gamma + \lambda$
   
   where $x = [\gamma|\lambda]$ and $|\lambda| = q$

3. If $x > m$ then $x \leftarrow x - m$
Algorithm 2
MCG Computation for Primes Close to $2^q$

1. $x \leftarrow \alpha \cdot x$

2. $x \leftarrow k \cdot \gamma + \lambda$

   where $x = [\gamma|\lambda]$ and $|\lambda| = q$

3. If $x > 2^q - 1$ then $x \leftarrow k \cdot \gamma' + \lambda'$

   where $x = [\gamma'|\lambda']$ and $|\lambda'| = q$

4. If $x > m$ then $x \leftarrow x - m$
Simulate Rolling Die $2^{29} \cdot 3$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm 1</td>
<td>11.0</td>
</tr>
<tr>
<td>Algorithm 2</td>
<td>13.2</td>
</tr>
<tr>
<td>lrand48()</td>
<td>32.4</td>
</tr>
<tr>
<td>drand48()</td>
<td>50.9</td>
</tr>
<tr>
<td>Modulus operation</td>
<td>58.9</td>
</tr>
</tbody>
</table>
Choose a Set of Multiplicative Constants

Find “primitive roots” of modulus $m$:

- What is a primitive root?
  \[ \{ \alpha^2, \alpha^3, \ldots, \alpha^{m-1} \} = \{ 2, 3, \ldots, m-1 \} \]
- How long does it take to find one?
  Fast to find a small one
- How to find others?
- Which ones to select?
  The ones that
  “give maximal period sequences of acceptable quality”
Finding Candidate Multipliers

- Apply LLL reduction
  - Approximation to Spectral Results
  - NTL Library
    - Fast implementation
- Run Empirical Tests
  - Four permutation tests
  - One uniformity test
  - Six independence tests
- Check Period
Example

MCG: \[ 26891986 x_n \mod 2^{33} - 9 \]

LLL-spectral result: 0.756007
Example

MCG: \[ 26891986 \times n \mod 2^{33} - 9 \]
LLL-spectral result: 0.756007
Example

MCG: $26891986 \times n \mod 2^{33} - 9$

LLL-spectral result: 0.756007
Example

MCG: \[ 8137022074 \times x_n \mod 2^{33} - 9 \]
LLL-spectral result: \[ 0.753160 \]
Example

MCG: \( 8137022074 \times_n \mod 2^{33}-9 \)

LLL-spectral result: 0.753160
MCG: $8137022074 x_n \mod 2^{33} - 9$

LLL-spectral result: $0.753160$
MCG: $8137022074 x_n \mod 2^{33} - 9$
LLL-spectral result: 0.753160
Checking Period

- Time-consuming task for large $2^q$
- Which algorithm is fastest?
  - Unsettled in the literature
  - Discovered modification to Brent’s algorithm
    - Finds exact period when it is small
    - Halts in a reasonable amount of time in worst case
    - If period is short, there is negligible effect on run-time
Using Brent’s Modified Algorithm

<table>
<thead>
<tr>
<th>MCG</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>$26891986 \ x_n \mod 2^{33}-9$</td>
<td>8589934582</td>
</tr>
<tr>
<td>LLL-spectral result: 0.756007</td>
<td></td>
</tr>
<tr>
<td>Passes Empirical Tests</td>
<td></td>
</tr>
<tr>
<td>$8137022074 \ x_n \mod 2^{33}-9$</td>
<td>19739</td>
</tr>
<tr>
<td>LLL-spectral result: 0.753160</td>
<td></td>
</tr>
<tr>
<td>Rejected</td>
<td></td>
</tr>
</tbody>
</table>
Calculating \( \pi \) in Parallel

Use MCG to pick \( 2^{32} = 1024^3 \) points from a cube and compute how many fall in the largest sphere inside the cube using 128 processors on 32 nodes with both MPI and hybrid MPI + OpenMP.

<table>
<thead>
<tr>
<th>Test</th>
<th>Approximation to ( \pi )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same multiplier, Different seed</td>
<td>3.1415769774466753</td>
</tr>
<tr>
<td>Different multiplier, Same seed</td>
<td>3.1415196494199336</td>
</tr>
</tbody>
</table>
Calculating \( \Pi \) in Parallel

Each process of 128 MPI processes uses an MCG with the same seed and different multipliers to pick 128 points from a cube. The points coincident with the largest sphere contained in the cube are collected from all processes to generate the following 3D plot.
Summary

- Demonstrated the performance benefit
- Fast Implementation of LLL reduction
- Shown that high LLL-spectral results are not enough
  - Proposed a new empirical testing procedure
  - Discovered a way to check the period
- Emphasized importance of running tests for each application
Acknowledgements

Thanks to Dr. Jeanie Osburn for her support.

This work was performed entirely on the Cray XD1 system at NRL-DC under the auspices of the U. S. Department of Defense (DoD) High Performance Computer Modernization Program (HPCMP).