Random Number Generation for Parallel and Threaded Programs

Gregory R. Warnes

March 22, 1999

Overall Goals

- Valid inference
- Reproducibility
- Efficiency

Necessary Features

- Non-overlapping
- Independent

Desirable Features

- Reproducibility
- Efficiency
- Simplicity

Current Language Approaches

Leave it to the User R, S, Splus, HPF, ZPL, ...

Single Random Number "Server" Java

Current User Approaches

Naive Users don't realize there is a problem == no change.

Knowledgeable R, S, Splus users use a different seed for each process.

High Performance Computing Community use offsets of a known separation in the stream.

Problems with Current User Approaches

Naive Users: No Change

- non-independent, often identical, random streams
- seriously jeopardizes validity of inference.

Problems with Current User Approaches

Knowledgeable Users: Separate seed for each process

- difficult to choose seeds that ensure independent and non-overlapping streams

Problems with Current User Approaches

HPC Community: Use offsets of known separation

- Known separation does not guarantee independence.
- Long simulations (fast computers) may require random numbers that exceed the separation.
- Not practical for all generators.

Problem with "Singe Source" (JAVA) Approach

- sequence obtained by each thread is non-deterministic
- depends on the exact order of calls among threads, which depends on non-deterministic system timings.

My Approach to Parallel Random Number Generation

Idea: Use a different random number generator for each process.

How: Use fixed generator form, with different "magic constants"

Implementation: Parallel form of Bruce Collings' (1987 JASA) random number generator.

Availability: C code.

Putative Properties of Parallel Collings Generator

- Independent streams
- Non-overlapping
- Reproducible (provided same number of processes!)