C/C++ Implementation of kinetic Monte Carlo

Peter Kratzer

Fritz-Haber-Institut der MPG, Berlin, Germany
Programming language: C / C++

useful features of C/C++

• Bit-wise logical operator
  – AND &
  – OR |
  – negate ~
• recursive function calls
• C++ object-oriented programming

topics of interest

• data representation
• random number generator
• periodic boundary conditions

example programs

• process-type-list algorithm: C program simple
• binary-tree algorithm: C++ program tree
Data representation

• SOS model:
  only 2-dimensional array required to store the local height of the surface
  int height[i][j]

• more complex models, e.g. with different species:
  3-dimensional array
  char lattice[i][j][h]
  #define A_species 0x001
  #define B_species 0x010
  #define C_species 0x100

  query
  if (lattice[i][j][h] & A_species) ...

  assignment
  lattice[i][j][h] |= B_species;

  removal
  lattice[i][j][h] &= ~C_species;
Data representation

- “multispin-coding” in Ising / lattice-gas models

Each bit in a data word addresses one node in a sub-lattice $\rightarrow$ quasi-parallel processing of many spins at a time

unsigned long
Random number generators

• frequently used types of random number generators
  – congruential generators (based on prime cosets spaces)
  – Knuth’s subtractive method \texttt{irnd}
  – bit-shifting polynomial twisters:
    Mersenne twister \texttt{mt19937}

• possible problems
  – repetition period possibly too short
  – single congruential generators have short-time correlations
    \textbf{solution}: interleaved congruential generators
  – real numbers from congruential generators: trailing digits are “less random” than leading digits
    \textbf{solution}: don’t use congruential generators if rate constants differ by orders of magnitude
  – risk of a “blind spot” in the random sequence, processes with very small rates are “overlooked”
    \textbf{solution}: “index-shuffling” in the rate list
Periodic boundary conditions

- modulo operations: possible, but slow
- indirect addressing
  \[ \text{height}[i-1][j] \rightarrow \text{height}[\text{BACK}[i]][j] \]

makePBCs()
{ for (j=1; j<L, j++) BACK[j] = (j-1+L)\%L;
  return; }

- use powers of 2 for the lattice dimensions, bit mask in address space
  \[ L = \text{pow}(\text{LBL}, 2); \]
  \[ \text{LL} = L-1; \]
  \[ \text{height}[i-1][j] \rightarrow \text{height}[((i-1) \& \text{LL})][j] \]
class Node{
public:
    Node(char d, char i, char j);
    int select(double rate, char* i, char* j);
    void set_rate(double input_rate);
    double update(void);
    double get_rate(void);
    ~Node();

protected:
    char depth, ix, jy;
    Node *westnode, *eastnode, *southnode, *northnode;
    double rate;
};

Recursive instancies of Node are used to build up the tree of total depth LBL.