Data Blocking

Jon K. Nilsen

Department of Physics and Scientific Computing Group
University of Oslo, N-0316 Oslo, Norway

Spring 2008

Outline

Data Blocking

- Implementing blocking
- Using `vmc_blocking`

Implementing blocking

vmc_blocking.cpp main()

```
int main (int nargs, char* args[])
{

  int n_procs, min_block_size, max_block_size, n_block_samples;
  //   Read from screen a possible new vaue of n
  if (nargs > 4) {
    n_procs         = atoi(args[1]);
    min_block_size  = atoi(args[2]);
    max_block_size  = atoi(args[3]);
    n_block_samples = atoi(args[4]);
  }
  else{
    cerr << "usage: ./vmc_blocking.x <n_procs> <min_bloc_size> "
         << "<max_block_size> <n_block_samples>" << endl;
    exit(1);
  }

  // get file size using stat
  struct stat result;
  int local_n,n;

  if(stat("blocks_rank0.dat", &result) == 0){
    local_n = result.st_size/sizeof(double);
    n = local_n*n_procs;
  }
  else{
    cerr << "error in getting file size" << endl;
    exit(1);
  }
```

Implementing blocking

vmc_blocking.cpp main()

```
// get all mc results from files
double* mc_results = new double[n];

for(int i=0; i<n_procs; i++){
  ostringstream ost;
  ost << "blocks_rank" << i << ".dat";
  ifstream infile;
  infile.open(ost.str().c_str(), ios::in | ios::binary);
  infile.read((char*)&(mc_results[i*local_n]),result.st_size);
  infile.close();
}

// and summarize
double mean, sigma;
double res[2];

meanvar(mc_results, n, res);

mean = res[0]; sigma= res[1];

// Open file for writing, writing results in formated output for plotting:
ofstream outfile;
outfile.open("blockres.dat", ios::out);

outfile << setprecision(10);
```

Implementing blocking

vmc_blocking.cpp main()

```cpp
  double* block_results = new double[n_block_samples];

  int block_size, block_step_length;

  block_step_length = (max_block_size-min_block_size)/n_block_samples;

  // loop over block sizes
  for(int i=0; i<n_block_samples; i++){
    block_size = min_block_size+i*block_step_length;
    blocking(mc_results, n, block_size, res);

    mean  = res[0];
    sigma = res[1];

    // formated output
    outfile << block_size << "\t" << mean << "\t"
            << sqrt(sigma/((n/block_size)-1.0))
            << endl;

  }

  outfile.close();

  return 0;
}
```

Implementing blocking

vmc_blocking.cpp blocking()

```cpp
// find mean and variance of blocks of size block_size.
// mean and variance are stored in res
void blocking(double *vals, int n_vals, int block_size, double *res){

  // note: integer division will waste some values
  int n_blocks = n_vals/block_size;

  double* block_vals = new double[n_blocks];

  for(int i=0; i<n_blocks; i++){
    block_vals[i] = mean(vals+i*block_size, block_size);
  }

  meanvar(block_vals, n_blocks, res);

  delete block_vals;
}
```

Implementing blocking

vmc_blocking.cpp meanvar()

```
// find mean of values in vals
double mean(double *vals, int n_vals){

  double m=0;
  for(int i=0; i<n_vals; i++){
    m+=vals[i];
  }

  return m/double(n_vals);
}

// calculate mean and variance of vals, results stored in res
void meanvar(double *vals, int n_vals, double *res){
  double m2=0, m=0, val;
  for(int i=0; i<n_vals; i++){
    val=vals[i];
    m+=val;
    m2+=val*val;
  }

  m  /= double(n_vals);
  m2 /= double(n_vals);

  res[0] = m;
  res[1] = m2-(m*m);

}
```

Usage

Compiling and runing `vmc_blocking.cpp`

- Prerequisites: The files `blocks_rank*.dat` from VMC simulation (e.g. from `vmc_para.cpp`)
- Compiling: `g++ -O3 -o vmc_blocking.x vmc_blocking.cpp` (note: not `mpicxx`!)
- Usage:
  - Copy `vmc_blocking.x` to directory that contains `blocks_rank*.dat` and change to that directory
  - Run: `./vmc_blocking.x <n_procs> <min_block_size> <max_block_size> <n_block_samples>`
  - Results are written to `blockres.dat` with block size in first column, mean in second column and std.dev. in third column
- `n_procs` is number of processors used in the VMC simulation. `min_block_size`, `max_block_size` and `n_block_samples` defines the range and resolution of the result file