Using Wilson Kernel Library for Blue Gene/Q

IBM Research - Tokyo

23 Jan. 2014

Overview of Wilson kernel library

- **Highly optimized kernel library**
  - Parallel SU(3) Wilson-Dirac operator
    - Supports Dirac/Chiral representations
    - Supports Even-Odd preconditioning
  - Local linear algebra routines for SU(3)
    - Linear algebra for spinor arrays
    - SU(3) gauge matrix multiplications

Data layout of spinor and gauge field

- **4x3 representation of spinor field**
  - Advantage in SIMDization on Quad FPU
  - Moving spins in the innermost of the data structure
- **Array of structures (AoS) data layout**
  - Spinor field
    - double _Complex w[s][c][Nx][Ny][Nz][Nt];
      - s=4, c=3, (Nx, Ny, Nz, Nt) are local lattice size
    - for Even-Odd
      - double _Complex w[s][c][Nx/2][Ny][Nz][Nt];
    - also conventional 3x4 representation is supported (use libbgqwilson3x4*.a)
  - Gauge field
    - double _Complex u[3][3][Nx][Ny][Nz][Nt][g];
      - g = 4
    - for Even-Odd
      - double _Complex u[3][3][Nx/2][Ny][Nz][Nt][2][g];

Using Wilson library

- **Header file**
  – #include ”bgqwilson.h”
- **Libraries**
  – libbgqwilsonsmp.a : SMP supported version
  – libbgqwilson.a : Single thread version
  – libbgqwilson3x4smp.a : 3x4 representation (slower), SMP
  – libbgqwilson3x4.a : 3x4, single thread
- **Linking libraries**
  – -lbgnet -lbgqwilsonsmp

Initializing library

- **BGWilson_Init**
  - Call this function at the top of the application once (calling from main function is recommended)
  - Please place this call after MPI_Init if you use MPI functions
    - Using BGNET instead of MPI is recommended for “flat-MPI” execution
    - Please set following environmental variables when using MPI
      - MUSPI_NUMINJFIFOS=16
      - MUSPI_NUMBATIDS=8
  - void BGWilson_Init(int Lx,int Ly,int Lz,int Lt,int Px,int Py,int Pz,int Pt);
    - Lx,Ly,Lz,Lt: are global lattice size for the application
    - Px,Py,Pz,Pt: are numbers of processors for each direction
      - So the local lattice size will be (Lx/Px, Ly/Py, Lz/Pz, Lt/Pt)
      - These settings can not be modified in the application

Wilson-Dirac operators

- **BGWilson_Mult**
  - For lexical site index
  - void BGWilson_Mult(void* pV,void* pU,void* pW,double kappa,int mode);
    - This function calculates, V = W - kappa*U(W)
    - V,W are spinor arrays, U is gauge array, pointer to the array
    - Dirac/Chiral representation is selected by “mode” parameter, select one of following definitions
      - BGWILSON_DIRAC
      - BGWILSON_CHIRAL

Wilson-Dirac operators

- **BGWilson_MultEO**
  - For Even-Odd
  - void BGWilson_MultEO(void* pV,void* pU,void* pW,double kappa,int ieo,int mode);
    - This function calculates, V = -kappa*U(W)
    - V,W are spinor arrays, U is gauge array, pointer to the array
      - V, W contains even or odd half of the spinor field
    - “ieo” is used to define the direction of the operation, select one of following definitions
      - BGWILSON_ODD_TO_EVEN (=0)
      - BGWILSON_EVEN_TO_ODD (=1)
    - Dirac/Chiral representation is selected by “mode” parameter, select one of following definitions
      - BGWILSON_DIRAC (=1)
      - BGWILSON_CHIRAL (=2)

Wilson-Dirac operators

- **BGWilson_MultAddEO**
  - For Even-Odd
  - void BGWilson_MultAddEO(void* pV,void* pU,void* pW,void* pA,double kappa,int ieo,int mode);
    - This function calculates, V = A - kappa*U(W)
    - V,W,A are spinor arrays, U is gauge array, pointer to the array
      - V, W, A contains even or odd half of the spinor field
    - “ieo” is used to define the direction of the operation, select one of following definitions
      - BGWILSON_ODD_TO_EVEN
      - BGWILSON_EVEN_TO_ODD
    - Dirac/Chiral representation is selected by “mode” parameter, select one of following definitions
      - BGWILSON_DIRAC
      - BGWILSON_CHIRAL

Wilson-Dirac operator for specific direction only

- **BGWilson_Mult_Dir, BGWilson_MultAdd_Dir**
  - For lexical site index
  - void BGWilson_Mult_Dir(void* pV,void* pU,void* pW,double kappa,int mode,int dim,int dir);
    - This function calculates, V = -kappa*U(W)
  - void BGWilson_MultAdd_Dir(void* pV,void* pA,void* pU,void* pW,double kappa,int mode,int dim,int dir);
    - This function calculates, V = A -kappa*U(W)

    - V,W,A are spinor arrays, U is gauge array, pointer to the array
    - Dirac/Chiral representation is selected by “mode” parameter, select one of following definitions
      - BGWILSON_DIRAC
      - BGWILSON_CHIRAL
    - Dimension and direction is specified in dim and dir parameter
      - dim: BGWILSON_X, BGWILSON_Y, BGWILSON_Z, BGWILSON_T
      - dir: BGWILSON_FORWARD, BGWILSON_BACKWARD or BGWILSON_BOTH_DIRECTION

Wilson-Dirac operator for specific direction only

- **BGWilson_MultEO_Dir, BGWilson_MultAddEO_Dir**
  - For even-odd
  - void BGWilson_Mult_Dir(void* pV,void* pU,void* pW,double kappa,int mode,int dim,int dir);
    - This function calculates, V = -kappa*U(W)
  - void BGWilson_MultAddEO_Dir(void* pV,void* pA,void* pU,void* pW,double kappa,int ieo,int mode,int dim,int dir);
    - This function calculates, V = A -kappa*U(W)

    - V,W,A are spinor arrays, U is gauge array, pointer to the array
    - Dirac/Chiral representation is selected by “mode” parameter, select one of following definitions
      - BGWILSON_DIRAC
      - BGWILSON_CHIRAL
    - “ieo” is used to define the direction of the operation, select one of following definitions
      - BGWILSON_ODD_TO_EVEN
      - BGWILSON_EVEN_TO_ODD
    - Dimension and direction is specified in dim and dir parameter
      - dim: BGWILSON_X, BGWILSON_Y, BGWILSON_Z, BGWILSON_T
      - dir: BGWILSON_FORWARD, BGWILSON_BACKWARD or BGWILSON_BOTH_DIRECTION

Site shift operation

- **void BGWilson_Shift_Dir(void* pDest,void* pSrc,int size,int dim,int dir);**
  - For lexical site index
  - Data structure (size bytes) on the lattice sites are shifted from pSrc to pDest in dim dimension to dir direction
- **void BGWilson_ShiftEO_Dir(void* pDest,void* pSrc,int size,int dim,int dir,int ieo);**
  - For Even-Odd
- **Dimension and direction is specified in dim and dir parameter**
  - dim: BGWILSON_X, BGWILSON_Y, BGWILSON_Z, BGWILSON_T
  - dir: BGWILSON_FORWARD, BGWILSON_BACKWARD or BGWILSON_BOTH_DIRECTION
- **“ieo” is used to define the side of pSrc and pDest for Even-odd**
  - BGWILSON_ODD_TO_EVEN
  - BGWILSON_EVEN_TO_ODD

SU3 Spinor-Gauge multiplication

- **void BGWilsonSU3_MultU_1D(void* pV,void* pU,void* pW,int n);**
- **void BGWilsonSU3_MultAddU_1D(void* pV,void* pA,void* pU,void* pW,int n);**
  - This function calculates V=U(W) or V=A+U(W) for length n, each arrays are defined as
    - double _Complex pV[n][3],pW[n][3], pA[n][3], pU[n][9];
- **void BGWilsonSU3_MultU_1D_S(void* pV,int offV,void* pU,int offU,void* pW,int offW,int n);**
- **void BGWilsonSU3_MultAddU_1D_S(void* pV,int offV,void* pA,int offA,void* pU,int offU,void* pW,int offW,int n);**
  - This function calculates V=U(W) or V=A+U(W) for stridded access of n sequence, each arrays are defined as
    - double _Complex pV[n*offV][3],pW[n*offW][3],pA[n*offA][3],pU[n*offU][9];
    - Each arrays are accessed using each stride

SU3 Spinor-Gauge multiplication

- **void BGWilsonSU3_MultUt_1D(void* pV,void* pU,void* pW,int n);**
- **void BGWilsonSU3_MultAddUt_1D(void* pV,void* pA,void* pU,void* pW,int n);**
  - This function calculates V=Ut(W) or V=A+Ut(W) for length n, each arrays are defined as
    - double _Complex pV[n][3],pW[n][3], pA[n][3], pU[n][9];
- **void BGWilsonSU3_MultUt_1D_S(void* pV,int offV,void* pU,int offU,void* pW,int offW,int n);**
- **void BGWilsonSU3_MultAddUt_1D_S(void* pV,int offV,void* pA,int offA,void* pU,int offU,void* pW,int offW,int n);**
  - This function calculates V=Ut(W) or V=A+Ut(W) for strided access of n sequence, each arrays are defined as
    - double _Complex pV[n*offV][3],pW[n*offW][3],pA[n*offA][3],pU[n*offU][9];
    - Each arrays are accessed using each stride

SU(3) Linear algebra routines

- **Setting constant value**
  - void BGWilsonLA_SetConst(void* pV,double a,int ns);
  - void BGWilsonLA_SetConst_S(void* pV,double a,int ns,int wOffset);
    - pV is a pointer to spinor field, a is a value to be set, ns is number of sites to be set
    - _S is strided version, wOffset is a stride shown in number of sites
- **Copy**
  - void BGWilsonLA_Equate(void* pV,void* pW,int ns);
  - void BGWilsonLA_Equate_S(void* pV,void* pW,int ns,int wOffset);
    - V = W

SU(3) Linear algebra routines

- **Multiplying a scalar value**
  - void BGWilsonLA_MultScalar(void* pV,void* pW,double s,int ns);
  - void BGWilsonLA_MultScalar_S(void* pV,void* pW,double s,int ns,int wOffset);
    - V = W * s
- **Multiplying a scalar value and adding**
  - void BGWilsonLA_MultAddScalar(void* pV,void* pW,double s,int ns);
  - void BGWilsonLA_MultAddScalar_S(void* pV,void* pW,double s,int ns,int wOffset);
    - V += W * s

SU(3) Linear algebra routines

- **Add 2 fields**
  - void BGWilsonLA_Add(void* pV,void* pW,int ns);
  - void BGWilsonLA_Add_S(void* pV,void* pW,int ns,int wOffset);
    - V += W
- **Subtract 2 fields**
  - void BGWilsonLA_Sub(void* pV,void* pW,int ns);
  - void BGWilsonLA_Sub_S(void* pV,void* pW,int ns,int wOffset);
    - V -= W

SU(3) Linear algebra routines

- **Add 2 fields and multiply a scalar**
  - void BGWilsonLA_Add_MultAddScalar(void* pV,void* pX,void* pY,double s,int ns);
  - void BGWilsonLA_Add_MultAddScalar_S(void* pV,void* pX,void* pY,double s,int ns,int wOffset);
    - V += (X+Y) * s
- **Multiply a scalar value and add fields**
  - void BGWilsonLA_MultScalar_Add(void* pV,void* pW,double s,int ns);
  - void BGWilsonLA_MultScalar_Add_S(void* pV,void* pW,double s,int ns,int wOffset);
    - V = V * s + W

SU(3) Linear algebra routines

- **AXPY**
  - void BGWilsonLA_AXPY(void* pV,void* pX,void* pY,double a,int ns);
  - void BGWilsonLA_AXPY_S(void* pV,void* pX,void* pY,double a,int ns,int wOffset);
    - V = a * X + Y
- **AXMY**
  - void BGWilsonLA_AXMY(void* pV,void* pX,void* pY,double a,int ns);
  - void BGWilsonLA_AXMY_S(void* pV,void* pX,void* pY,double a,int ns,int wOffset);
    - V = a * X - Y

SU(3) Linear algebra routines

- **AXPBY**
  - void BGWilsonLA_AXPBY(void* pV,void* pX,void* pY,double a,double b,int ns);
  - void BGWilsonLA_AXPBY_S(void* pV,void* pX,void* pY,double a,double b,int ns,int wOffset);
    - V = a * X + b * Y
- **AXPBYPZ**
  - void BGWilsonLA_AXPBYPZ(void* pV,void* pX,void* pY,void* pZ,double a,double b,int ns);
  - void BGWilsonLA_AXPBYPZ_S(void* pV,void* pX,void* pY,void* pZ,double a,double b,int ns,int wOffset);
    - V = a * X + b * Y + Z

SU(3) Linear algebra routines

- **Norm of array**
  - void BGWilsonLA_Norm(double* AV,void* pV,int ns);
  - void BGWilsonLA_Norm_S(double* AV,void* pV,int ns,int wOffset);
    - AV = V*V
      - V is treated as double array inside this function (V*V is not complex multiplication, 24 double elements in one spinor)
- **Dot product**
  - void BGWilsonLA_DotProd(double* AV,void* pV,void* pW,int ns);
  - void BGWilsonLA_DotProd_S(double* AV,void* pV,void* pW,int ns,int wOffset);
    - AV = V*W
      - V and W are treated as double array inside this function (V*W is not complex multiplication, 24 double elements in one spinor)

SU(3) Linear algebra routines

- **AXPY and norm of array**
  - void BGWilsonLA_AXPY_Norm(void* pV,double* AV,void* pX,void* pY,double a,int ns);
  - void BGWilsonLA_AXPY_Norm_S(void* pV,double* AV,void* pX,void* pY,double a,int ns,int wOffset);
    - V = a * X + Y
    - AV = V*V
- **Norm and dot product of arrays**
  - void BGWilsonLA_Norm_DotProd(double* pAN,double* pAV,void* pV,void* pW,int ns);
  - void BGWilsonLA_Norm_DotProd_S(double* pAN,double* pAV,void* pV,void* pW,int ns,int wOffset);
    - AV = V*V
    - AN = V*W

SU(3) Linear algebra routines

- **Mutiply a scala and add and norm of array**
  - void BGWilsonLA_MultAddScalar_Norm(void* pV,double* AV,void* pW,double s,int ns);
  - void BGWilsonLA_MultAddScalar_Norm_S(void* pV,double* AV,void* pW,double s,int ns,int wOffset);
    - V += s * W
    - AV = V*V

SU(3) gauge matrix linear algebra routines

- **Copy**
  - void BGWilsonSU3_MatEquate(void* pV,void* pW,int ns);
    - V = W
- **Set Zero**
  - void BGWilsonSU3_MatZero(void* pV,int ns);
    - V = 0.0
- **Set 1**
  - void BGWilsonSU3_MatUnity(void* pV,int ns);
    - V = 1.0

SU(3) gauge matrix linear algebra routines

- **Add**
  - void BGWilsonSU3_MatAdd(void* pV,void* pW,int ns);
    - V = V + W
  - void BGWilsonSU3_MatAdd_ND(void* pV,void* pW,int ns);
    - $V = V + W^t$
- **Sub**
  - void BGWilsonSU3_MatSub(void* pV,void* pW,int ns);
    - V = V - W
- **Multiply scalar**
  - void BGWilsonSU3_MatMultScalar(void* pV,double a,int ns);
    - V = a * V

SU(3) gauge matrix multiplication routines

- **Multiply 2 matrix arrays**
  - void BGWilsonSU3_MatMult_NN(void* pV,void* pA,void* pB,int n);
    - Calculates V = A * B
    - pV, pA, pB are pointer to the gauge array, n is number of SU(3) matrices to be multiplied sequentially
  - void BGWilsonSU3_MatMult_ND(void* pV,void* pA,void* pB,int n);
    - $V = A * B^t$
  - void BGWilsonSU3_MatMult_DN(void* pV,void* pA,void* pB,int n);
    - $V = A^t * B$
  - void BGWilsonSU3_MatMult_DD(void* pV,void* pA,void* pB,int n);
    - $V = A^t * B^t$

SU(3) gauge matrix multiplication routines

- **Multiply 2 matrix arrays and add**
  - void BGWilsonSU3_MatMultAdd_NN(void* pVOut,void* pVIn,void* pA,void* pB,int n);
    - Calculates VOut = VIn + A * B
    - pVOut, pVIn, pA, pB are pointer to the gauge array, n is number of SU(3) matrices to be multiplied sequentially
  - void BGWilsonSU3_MatMultAdd_ND(void* pVOut,void* pVIn,void* pA,void* pB,int n);
    - VOut = VIn + A * B^t
  - void BGWilsonSU3_MatMultAdd_DN(void* pVOut,void* pVIn,void* pA,void* pB,int n);
    - VOut = VIn + A^t * B
  - void BGWilsonSU3_MatMultAdd_DD(void* pVOut,void* pVIn,void* pA,void* pB,int n);
    - VOut = VIn + A^t * B^t

SU(3) gauge matrix multiplication routines

- **Multiply 3 matrix arrays**
  - void BGWilsonSU3_MatMult_NND(void* pV ,void* pA,void* pB,void* pC,int n);
    - Calculates $V = A * B * C^t$
    - pV, pA, pB, pC are pointer to the gauge array, n is number of SU(3) matrices to be multiplied sequentially
  - void BGWilsonSU3_MatMult_DNN(void* pV,void* pA,void* pB,void* pC,int n);
    - Calculates $V = A^t * B * C$

Using shared memory parallelization by OpenMP

- **Put parallel sections outside of the Wilson library functions**
  - All the functions are thread-safe, but there is no parallel section inside functions to avoid overhead of starting threads
- **Set the number of threads to be used for parallelization**
  - OMP_NUM_THREADS environmental variable
  - omp_set_num_threads function

OpenMP example

```
#pragma omp parallel
   {
      int tid,nid,is,ns,ieo;
      nid = omp_get_num_threads();
      tid = omp_get_thread_num();
      is = qcdNsite * tid / nid;
      ns = qcdNsite * (tid + 1) / nid - is;

      BGWilsonLA_Equate(pS + is,pB + is,ns);
      BGWilsonLA_Equate(pM + is,pB + qcdNsite + is,ns);

      ieo = 1;
      DMultEO(pR,pU,pM,mCks,ieo,mode);
      BGWilsonLA_Sub(pS + is,pR + is,ns);

      BGWilsonLA_Equate(pR + is,pS + is,ns);
      BGWilsonLA_Equate(pX + is,pS + is,ns);

      ieo = 2;
      DMultEO(pM,pU,pS,mCks,ieo,mode);
      ieo = 1;
      DMultAddEO(pP,pU,pM,pS,Cks,ieo,mode);
```

extent for this thread is calculated here

part of array is calculated in linear algebra functions

whole of arrays are passed to Wilson functions, extent is calculated inside