Lattice QCD simulations on SuperMUC and beyond

Hinnerk Stüben

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

SuperMUC Status and Results Workshop

Leibniz-Rechenzentrum, Garching

July 9, 2014

Overview

- supercomputing aspects of Lattice QCD simulations
- QCD program: BQCD
- SIMD vectorisation
- future performance expectations

BQCD in production

- can simulate $N_f = 2 + 1$ fermion flavours coupled to QED
- authors: H.S. and Yoshifumi Nakamura (RIKEN, Japan)
- used by the QCDSF collaboration
  - on supercomputers at LRZ (project h006z)
    - HLRB-I: Hitachi SR8000 (2001-2006)
    - HLRB-II: SGI Altix 4700 (2007-2011)
    - SuperMUC: IBM iDataPlex (2012)
- and lattice QCD groups at Universität Regensburg

BQCD benchmark

- fhe $N_f = 2$ version is a well known supercomputer benchmark
- the benchmark is the *conjugate gradient* solver
- benchmark suites containing BQCD
  - LRZ
    - HLRB-II
    - SuperMUC
  - North German Supercomputing Alliance (HLRN)
    - HLRN-I
    - HLRN-II
    - HLRN-III
  - PRACE

BQCD benchmark

- scales to large numbers of cores (at least 300.000)
- communication intensive (tests the network)
- can also be used for benchmarking fat nodes (pure OpenMP mode)
- GPU computing
  (a multi-GPU version of the kernel was implemented by Mike Clark, NVIDIA)

Performance

- example
  - lattice: $48^3 \times 96$
  - decomposition: $1 \times 16 \times 12 \times 48$ processes (9216 cores)
  - lattice per core: $48 \times 3 \times 4 \times 2$ (1152 sites)
  - performance: 17.3 TFlop/s (1880 MFlop/s per core)
  - machine: Cray XC30 (HLRN-III, phase 1)

- discussion
  - local volume is small
    - $\rightarrow$ super-linear speedup
    - $\rightarrow$ data caches play a roll
    - $\rightarrow$ try to improve performance be employing SIMD

SIMD units

- processing of a loop

```
for i := 1 to 100 do
     c[i] := a[i] + b[i]
```

sequential processing

for i := 1 to 100 do

a[i]
+
b[i]
=
c[i]

SIMD processing

for i := 1 to 100 **step 4** do

a[i]	a[i+1]	a[i+2]	a[i+3]

+

b[i]	b[i+1]	b[i+2]	b[i+3]

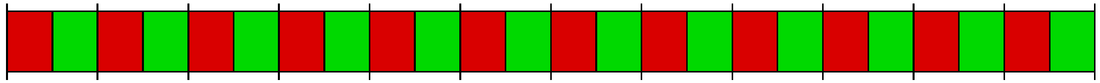
=

c[i]	c[i+1]	c[i+2]	c[i+3]

Layout of complex arrays/vectors

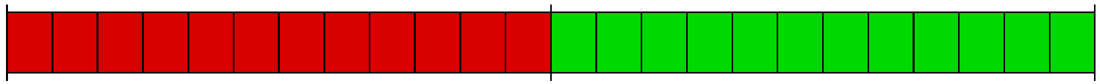
- standard (micro-processors)

`z(re:im, n)`



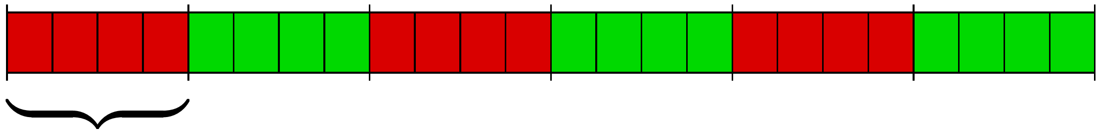
- vector computers

`z(n, re:im)`



- SIMD vectors

`z(B, re:im, n/B)`



B: width of a SIMD unit

Programming loops

- **standard**

```
complex(8), dimension(n) :: a, b, c
do i = 1, n
   a(i) = b(i) + c(i)
enddo
```

- **'vector'**

```
real(8), dimension(n, re:im) :: a, b, c
do i = 1, n
   a(i, re) = b(i, re) + c(i, re)
   a(i, im) = b(i, im) + c(i, im)
enddo
```

- **'SIMD ready'**

```
real(8), dimension(4, re:im, n/4) :: a, b, c
do i = 1, n, 4
   a(:, re, i) = b(:, re, i) + c(:, re, i)
   a(:, im, i) = b(:, im, i) + c(:, im, i)
enddo
```

- **SIMD intrinsics**

```
real(8), dimension(4, re:im, n/4) :: a, b, c
do i = 1, n, 4
   Store(a(re, i)), Add(Load(b(re, i)), Load(c(re, i))
   Store(a(im, i)), Add(Load(b(im, i)), Load(c(im, i))
enddo
```

What the compiler does with typical QCD loops

loop	implementation	performance [MFlop/s per core] L2 cache	memory
$\begin{pmatrix} \bullet \\ \bullet \\ \bullet \end{pmatrix}_i += \begin{pmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{pmatrix}_i \times \begin{pmatrix} \bullet \\ \bullet \\ \bullet \end{pmatrix}_i$	standard	5670	880
	'vector'	1820	930
	'SIMD ready'	9930 $\leftarrow$	990
	SIMD intrinsics	9240	900
$\begin{pmatrix} \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \end{pmatrix}_i += \begin{pmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{pmatrix}_i \times \begin{pmatrix} \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \end{pmatrix}_i$	standard	6230	1260
	'vector'	2240	1290
	'SIMD ready'	2290 $\leftarrow$	950
	SIMD intrinsics	10200	1270

- Intel compiler 14.0.2, flags: `-O3 -mavx`
- run on all cores of a HLRN-III node (2×12 cores)
- complex arithmetic, loop length $= 512$ (cache is cleared in the *memory* case)

Points to care about

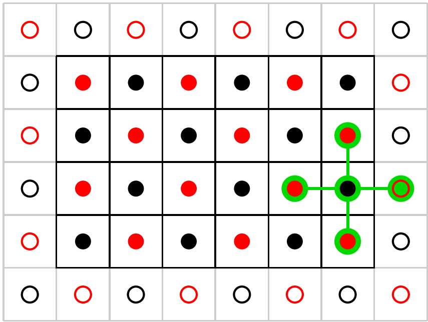
The SIMD approach affects all parallelisation levels:

- loop body
  - neighbour access
- OpenMP
  - work sharing
- MPI
  - domain decomposition

Two implementation details

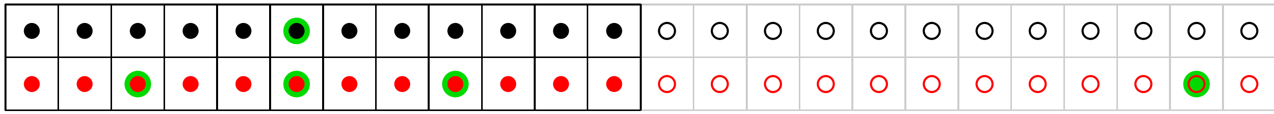
- data layout in BQCD (4 dimensions, chessboard decomposition, 9-point stencil)

lattice geometry



⟶

computer memory



- loops in BQCD (run over one colour and are collapsed)

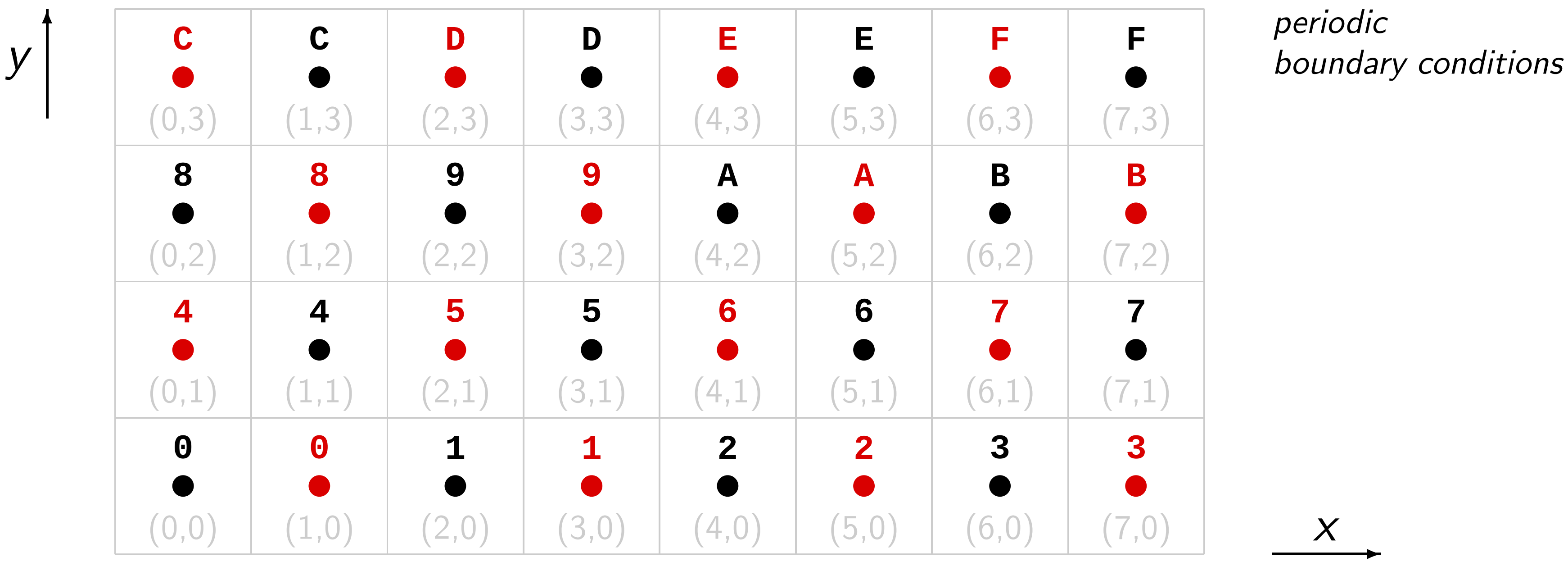
```
do t = 1, Nt
   do z = 1, Nz
      do y = 1, Ny
         do x = 1, Nx/2
            ...
```

⟶

```
do i = 1, vol/2
   ...
```

⇝ neighbour access (→next slide)

Chessboard decomposition in two (and more) dimensions



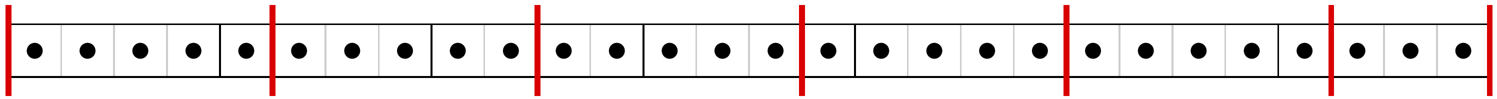
x-direction			*y*-direction		
sites	neighbours forward	neighbours backward	sites	neighbours forward	neighbours backward
0123	**0123**	*3012*	**0123**	**4567**	**CDEF**
4567	*5674*	**4567**	**4567**	**89AB**	**0123**
89AB	**89AB**	*B89A*	**89AB**	**CDEF**	**4567**
CDEF	*DEFC*	**CDEF**	**CDEF**	**0123**	**89AB**

SIMD and OpenMP

- OpenMP work sharing might destroy *alignment*

```
!$omp parallel do schedule(static)
do i = 1, 28
   y(i) = ...
enddo
```

OMP_NUM_THREADS=**6**



→ simd construct in OpenMP 4.0

- no problem for 'OpenMP ready' loops

SIMD and domain decompositions in BQCD

- recall (implicit) loop structure (`Nx, Ny, Nz, Nt` might be global or local lattice dimensions):

```
do t = 1, Nt                  →          do i = 1, vol/2
   do z = 1, Nz                             ...
      do y = 1, Ny
         do x = 1, Nx/2
            ...
```

- globally `Nx = Ny = Nz = Nspace` and `Nt = Ntime = 2` $\times$ `Nspace`

- SIMD view
  - `Nspace/2` might not be a multiple of the *simd width*

  → choose `t` to be the fastest running dimension

- MPI view
  - boundaries in the slowest running dimension are optimal for MPI (are consecutive in memory)

  → choose the longest dimension to be the slowest running dimension

Discussion

- general aspects
  - in future it might suffice to program 'SIMD ready' (hopefully)
  - still, SIMDisation might require other conceptual changes
- BQCD specific
  - in the super-linear scaling region, about half of the time is spent in MPI communication
  - if compute performance is improved, MPI communication should be accelerated as well
    - remote direct memory access (RDMA)
    - overlapping communication and computation

Future performance expectations

- benchmark performance and machine size expectations
  (assumption: sustained performance per core is constant)

	starting point	next lattice	nice to have
lattice size	$48^3 \times 96$	$64^3 \times 128$	$96^3 \times 192$
#cores	9.000	32.000	145.000
performance	17.3 TF	60 TF	220 TF

- machine usage model
  - on a machine with a hierarchical network it could be that good performance at high core counts can only be achieved on certain partitions
  - in that situation block times would be preferable over scheduling individual jobs