SLDA and TD-SLDA on JaguarPF

K. J. Roche
High Performance Computing Group
Pacific Northwest National Laboratory
Nuclear Theory Group
University of Washington

In collaboration with:
Aurel Bulgac
Michael Forbes
Alan Luo
Piotr Magierski
Ionel Stetcu
Sukjin Yoon
Yongle Yu
## JaguarPF and Trends in Target Computing Platforms

<table>
<thead>
<tr>
<th>Hex-Core AMD Opteron (TM)</th>
<th>2.6e9 Hz clock</th>
<th>4 FP_OPs / cycle / core 128 bit registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEs</td>
<td>18,688 nodes</td>
<td>224,256 cpu-cores (processors)</td>
</tr>
<tr>
<td>Memory</td>
<td>16 GB / node</td>
<td>dual socket nodes</td>
</tr>
<tr>
<td></td>
<td>6 MB shared L3 / chip</td>
<td>800 MHz DDR2 DIMM</td>
</tr>
<tr>
<td></td>
<td>512 KB L2 / core</td>
<td>25.6 GBps / node memory bw</td>
</tr>
<tr>
<td></td>
<td>64 KB D,I L1 / core</td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>AMD HT</td>
<td>3D torus topology</td>
</tr>
<tr>
<td></td>
<td>SeaStar2+</td>
<td>6 switch ports / SeaStar2+ chip</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9.6 GBps interconnect bw / port</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2GBps injection bw</td>
</tr>
<tr>
<td>Operating Systems</td>
<td>Cray Linux Environment (CLE) (xt-os2.2.41A)</td>
<td>SuSE Linux on service / io nodes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FY</th>
<th>Aggregated Cycles</th>
<th>Aggregated Memory</th>
<th>Aggregated FLOPs</th>
<th>Memory/FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td>65.7888 THz</td>
<td>61.1875 TB</td>
<td>263.155 TF</td>
<td>0.2556</td>
</tr>
<tr>
<td>2009</td>
<td>343.8592 THz</td>
<td>321.057 TB</td>
<td>1.375 PF</td>
<td>0.2567</td>
</tr>
<tr>
<td>2010</td>
<td>583.0656 THz</td>
<td>321.057 TB</td>
<td>2.332 PF</td>
<td>0.1513</td>
</tr>
</tbody>
</table>
Structure of the SLDA Production Software

Init Physics → Solver

- stationary code specific
- time dependent code
- observable data

Evolve Time

- solver input
- observables(t)

post analysis, learn, refine

DISK

Solver

- learn, refine
- n.conv
- conv

DISK

- wave functions, potentials (gases), some scalars
- potentials, optimization info

DISK

observables(t)

Halt

- time dependent wave functions, potentials (gases), some scalars

 restarting

observables(t)

Tuesday, June 22, 2010
A Small-scale DEMO of the Tool : Unitary Fermi Gas in Trap + Ball and Rod Stirring

300 particles ; 32^3 lattice ; 104,132 time steps ; 1509 I/O events of analysis data

Run 1 self-consistent solver generated stationary solutions for the system

Run 2 initialized the TD code, executed a total 87,271 time steps, 1264 completed I/O events, check pointed

Run 3 restarted at time step 87,272, executed 16,861 additional time steps, 245 additional I/O events, exited cleanly

<table>
<thead>
<tr>
<th></th>
<th>Run 1</th>
<th>Run 2</th>
<th>Run 3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEs</td>
<td>612</td>
<td>9458</td>
<td>9458</td>
<td>46,381.02</td>
</tr>
<tr>
<td>Time(s)</td>
<td>451.081</td>
<td>34,909.812</td>
<td>11,020.127</td>
<td>120,744.8401</td>
</tr>
<tr>
<td>CPU $(hours)$</td>
<td>76.68377</td>
<td>91,715.83386</td>
<td>28,952.32255</td>
<td>120,744.8401</td>
</tr>
</tbody>
</table>

Energy vs time

Convergence Error vs Iteration
Numerical Studies of Vortex Formation and Dynamics in Superfluid Fermi Systems
Benchmark(1) on JaguarPF for ASCR’s OMB PART
Software Effectiveness Metric: UG Problem

5216 particles; 103,917 wavefunctions; 50x50x100 lattice; 2,051 (100K) time steps; 26 I/O events of analysis data

Solver (Run1)
- 51 parallel, parallel groups
- 144 PEs / group
- simultaneous vs in sequence, perfect strong scaling
- 129 iterations to converge

TD (Run2)
- initialize (read soln) TD code
- 1 time step
- 1 data analysis io event
- checkpoint TD wavefunctions
  - 8TB data written
  - 24 Lustre write groups

TD (Run3)
- restart TD code
- 2050 time steps
- 25 data analysis io events
- clean exit

<table>
<thead>
<tr>
<th>Machine Data</th>
<th>Run 1</th>
<th>Run 2</th>
<th>Run 3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructions</td>
<td>3.335083409e17</td>
<td>3.538162667e18</td>
<td>4.415760819e18</td>
<td>8.287431827e18</td>
</tr>
<tr>
<td>Floating Point Ops</td>
<td>3.628460357e15</td>
<td>1.91882289e14</td>
<td>3.235240267e17</td>
<td>3.272443593e17</td>
</tr>
<tr>
<td>Wall Time(s)</td>
<td>11,085.29975</td>
<td>8,291.248311</td>
<td>12,913.89644</td>
<td>32,290.4445</td>
</tr>
<tr>
<td>CPU $ (hours)</td>
<td>22,614.01149</td>
<td>239,333.7919</td>
<td>372,770.3823</td>
<td>634,718.1857</td>
</tr>
<tr>
<td>PEs</td>
<td>7,344</td>
<td>103,917</td>
<td>103,917</td>
<td></td>
</tr>
</tbody>
</table>

In Q3, successfully benchmarked the TD code on 217,752 2-component qpwf system on 62^3 lattice. - over 97% of the complete system!
Users Beware ... Some Learned Lessons

Aside on FILEs and IO

ANSI C
• stream of BYTES
• points to a FILE structure
• fopen, fwrite, fread, fclose

Fortran
• sequence of records
• open, write, read, close
• IOLENGTH, RECL

```c
typedef struct {
    int level;    /* fill/empty level of buffer */
    unsigned flags;   /* File status flags */
    char fd;        /* File descriptor */
    unsigned char hold; /* Ungetc char if no buffer */
    int bsize;      /* Buffer size */
    unsigned char *buffer; /* Data transfer buffer */
    unsigned char *curp; /* Current active pointer */
    unsigned istemp; /* Temporary file indicator */
    short token;    /* Used for validity checking */
} FILE;
```

```c
void f_copn_ ( char * ffn , int * ffd , int * len ) ;
void f_ccls_ ( int * ffd ) ;
void f.crm_ ( char * ffn , int * len ) ;
void f_cwr_ ( int * ffd , void * fbf , int * fsz , int * nobj , int * ierr ) ;
void f_crd_ ( int * ffd , void * fbf , int * fsz , int * nobj , int * ierr ) ;
```

```c
fn = '/tmp/work/roche/mpt-omp/ben.txt'
CHAR(0)
call f_copn ( fn , fd , LEN( fn ) )
call f_cwr ( fd , a , 16 , ndim , ierr )
call f_ccls ( fd )
call f_copn ( fn , fd , LEN( fn ) )
call f_crd ( fd , a_bk , 16 , ndim , ierr )
call f_ccls ( fd )
call f.crm ( fn , LEN( fn ) )
```

Tuesday, June 22, 2010
Aside on FILEs and IO (2)

POSIX (UNIX)
- stream of BYTES
- file descriptors
  - index into file descriptor table
  - kept in user process
    - points to entry in system in-memory
      inode table
- open, write, read, close, ioctl

Spider (Lustre):
- MDS, file names and directories in the filesystem, file open, close, state mgt
- OSS, provides file service, and network request handling for set of OSTs
- OST, stores chunks of files as data objects - may be stripped across one or more OSTs
  - Spider has 672 OSTs
  - 7 TB per OST
  - 1 MB Default stripe size
  - 4 Default OST count
Aside on FILEs and IO (3) - SLDA Approach

module load liblut ; -LUT
lut___open() ;
lut___close() ;
lut_putl() ;
pwrite() ;
pread() ;

- form modulo classes from MPI communicator over the number of I/O groups
  - for both proton and neutron communicators in nuclear case (44 for protons, 44 for neutrons)

- fit the stripe size to the largest single data item if possible
  - eg for nuclear code and \(32^3\) lattice, a single 4-component term is \(4 \times 32^3 \times 16 / 2^{20} = 2\) MB

- set the stripe pattern (I use round-robin) and number of target OSTs (I use 88 in nuc code) for target PATH / FILE
  - eg lfs setstripe /tmp/work/roche/kio -s 2m -i -1 -c 88

Dominant I/O Demands for Checkpoint / Restart of Software:

- **unitary** : \(22 \times NWF \times Nx \times Ny \times Nz \times \text{sizeof(double complex)}\)
  - eg \(22 \times 103917 \times 50 \times 50 \times 100 \times 16 \approx 8,517\) GB or \(~8.317\) TB

- **nuclear** : \(44 \times NWF \times Nx \times Ny \times Nz \times \text{sizeof(double complex)}\)
  - eg \(44 \times 43366 \times 32^3 \times 16 \approx 931.691\) GB

**Performance:** POSIX ~ [225,350] MBps, my use of LUSTRE ~ [5,10] GBps
Another DEMO of the Tool: Nuclear LACM

280Cf; 32^3 lattice; 43,380 4-Component QPWFs, 13,173 time steps; 1317 I/O events of analysis data

- **Run 1** initialized the TD code, executed a total **10,951** time steps, **1095** completed I/O events, check pointed

- **Run 2** restarted at time step **10,952**, executed **2,121** additional time steps, **212** additional I/O events, check pointed

- **Run 3** restarted at time step **13,073**, executed **101** additional time steps, **10** additional I/O events, exited cleanly

Also successfully benchmarked the TD nuclear code w/ 130,098 4-component qpwfs on 32^3 to prepare for 32x32x50 LACM run with 83,260 4-component qpwfs.
Numerical Studies of Large Amplitude Collective Motion in Nuclear Systems

280Cf responds to quadropole excitation (visit aurel’s talk):
Effective Use of the Biggest Open Science Supercomputer in the World Today ... What Comes NEXT???

- finish FY10 Joule OMB PART exercise
- rewrite nuclear codes entirely in C
- further parallelization
  - expose the u and v components of the qpwfs
  - task oriented threading, streaming vectorized regions to GPU
    - not likely to get performance through compiler directives alone
- more i/o testing (if Lustre goes away, then what?)
  - don’t see this coming but ... should at least support GPFS
- wide area data movement
Thank You.