CP2K Parallelisation and Optimisation

CP2K Summer School, 19-22 June 2018

Iain Bethune

iain.bethune@stfc.ac.uk

@iainbethune
Overview

• Overview of Parallel Programming models
  • Shared Memory
  • Distributed Memory

• CP2K Algorithms and Data Structures

• Parallel Performance

• CP2K Timing Report
Parallel Programming Models

- Why do we need parallelism at all?
  - Parallel programming is (even) harder than sequential programming

- Single processors are reaching limitations
  - Clock rate stalled at ~2.5 GHz (due to heat)
  - Full benefits of vectorisation (SIMD) can be hard to realise
  - Chip vendors focused on low-power (for mobile devices)
Parallel Programming Models

• But we need more speed!
  • Solve problems faster (strong scaling)
  • Solve bigger problems in same time (weak scaling)
  • Tackle new science that emerges at long runtimes / large system size
  • Enables more accurate force models (HFX, MP2, RPA ...)

• Need strategies to split up our computation between different processors

• Ideally our program should run P times faster on P processors - but not in practice!
  • Some parts may be inherently serial (Amdahl’s Law)
  • Parallelisation will introduce overheads e.g. communication, load imbalance, synchronisation...
“The performance improvement to be gained by parallelisation is limited by the proportion of the code which is serial”

Gene Amdahl, 1967
Almost all modern CPUs are multi-core
- 2, 4, 6... CPU cores, sharing access to a common memory

This is Shared Memory Parallelism
- Several processors executing the same program
- Sharing the same address space i.e. the same variables
- Each processor runs a single ‘thread’
- Threads communicate by reading/writing to shared data

Example programming models include:
- OpenMP, POSIX threads (pthreads)
One very large whiteboard in a two-person office
  - the shared memory
Two people working on the same problem
  - the threads running on different cores attached to the memory

How do they collaborate?
  - working together
  - but not interfering

Also need *private* data
Hardware

Needs support from a shared-memory architecture

Memory

Shared Bus

Processor  Processor  Processor  Processor  Processor
Parallel Programming Models

- Most supercomputers are built from 1000s of nodes
  - Each node consists of some CPUs and memory
  - Connected together via a network

- This is Distributed Memory Parallelism
  - Several processors executing (usually) the same program
  - Each processor has its own address space
  - Each processor runs a single ‘process’
  - Threads communicate by passing messages

- Example programming models include:
  - MPI, SHMEM
Analogy

- Two whiteboards in different single-person offices
  - the distributed memory
- Two people working on the same problem
  - the processes on different nodes attached to the interconnect
- How do they collaborate?
  - to work on single problem
- Explicit communication
  - e.g. by telephone
  - no shared data
Hardware

• Natural map to distributed-memory
  • one process per processor-core
  • messages go over the interconnect, between nodes/OS’s
Parallel Programming Models

• CP2K can use OpenMP or MPI (ssmp or popt)
  • Use OpenMP for desktop PCs with multi-cores or
  • MPI for clusters and supercomputers
  • Maybe also support for Accelerators (GPUs)

• May also combine MPI and OpenMP (psmp)
  • Called hybrid or mixed-mode parallelism
  • Use shared memory within a node (with several processors)
  • Use message passing between nodes
  • Usually only useful for scaling to 10,000s of cores!
CP2K Algorithms and Data Structures

- (A,G) – distributed matrices
- (B,F) – realspace multigrids
- (C,E) – realspace data on planewave multigrids
- (D) – planewave grids
- (I,VI) – integration/collocation of gaussian products
- (II,V) – realspace-to-planewave transfer
- (III,IV) – FFTs (planewave transfer)
CP2K Algorithms and Data Structures

• Distributed realspace grids
  • Overcome memory bottleneck
  • Reduce communication costs
  • Parallel load balancing
    • On a single grid level
    • Re-ordering multiple grid levels
    • Finely balance with replicated tasks

<table>
<thead>
<tr>
<th>Level 1, fine grid, distributed</th>
<th>Level 2, medium grid, distributed</th>
<th>Level 3, coarse grid, replicated</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>5 6 8</td>
<td></td>
</tr>
<tr>
<td>4 5 6</td>
<td>3 1 7</td>
<td></td>
</tr>
<tr>
<td>7 8 9</td>
<td>9 4 2</td>
<td></td>
</tr>
</tbody>
</table>

- libgrid for optimised collocate/integrate routines
- ~5-10% speedup typical
CP2K Algorithms and Data Structures

- Fast Fourier Transforms
  - 1D or 2D decomposition
  - FFTW3 and CuFFT library interface
  - Cache and re-use data
    - FFTW plans, cartesian communicators
- DBCSR
  - Distributed MM based on Cannon’s Algorithm
  - Local multiplication recursive, cache oblivious
- Run on a square number of processes
- \texttt{lib[x]smm} for small block multiplications

- **GLOBAL\%FFT\_PLAN\_TYPE MEASURE | PATIENT**
  - Up to 5% Speedup possible

---

**Figure 5:** Comparing performance of SMM and Libsci BLAS for block sizes up to 22,22,22.
CP2K Algorithms and Data Structures

- **OpenMP**
  - Now in all key areas of CP2K
  - FFT, DBCSR, Collocate/Integrate,
  - Incremental addition over time
  - 1, 2 or 4 threads per process

- **Dense Linear Algebra**
  - Matrix operations during SCF
  - GEMM - ScalAPACK
  - SYEVD – ScalAPACK / ELPA

- To enable:
  - `GLOBAL%PREFERRED_DIAG_LIBRARY = ELPA`
  - `~D_ELPA=YYYYMM and link library`
  - Up to ~5x Speedup for large, metallic systems
Parallel Performance

- Different ways of comparing time-to-solution and compute resource...

- Speedup: $S = \frac{T_{ref}}{T_{par}}$

- Efficiency: $E_p = \frac{S_p}{p}$, ‘good’ scaling is $E > 0.7$

- If $E < 1$, then using more processors uses more compute time (AUs)

- Compromise between overall speed of calculation and efficient use of budget
  - Depends if you have one large or many smaller calculations
Parallel Performance: H2O-xx

Time per MD step (seconds)

Number of cores
Parallel Performance: LiH-HFX

Performance comparison of the LiH-HFX benchmark
Parallel Performance: H2O-LS-DFT

Parallel Performance: H2O-64-RI-MP2

![Graph showing parallel performance comparison between different architectures.](image-url)

- Time (seconds) on the y-axis.
- Number of nodes used on the x-axis.
- Points and lines indicate performance on various architectures and node counts.

ARCHER
HECToR Phase 3
Parallel Performance: GPUs

- ~25% speedup
- Only for DBCSR
CP2K Timing Report

- CP2K measures are reports time spent in routines and communication
- Timing reports are printed at the end of the run

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>CALLS</th>
<th>TOT TIME [s]</th>
<th>AVE VOLUME [Bytes]</th>
<th>PERFORMANCE [MB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP_Group</td>
<td>4</td>
<td>0.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MP_Bcast</td>
<td>186</td>
<td>0.018</td>
<td>958318.</td>
<td>9942.82</td>
</tr>
<tr>
<td>MP_Allreduce</td>
<td>1418</td>
<td>0.619</td>
<td>2239.</td>
<td>5.13</td>
</tr>
<tr>
<td>MP_Gather</td>
<td>44</td>
<td>0.321</td>
<td>21504.</td>
<td>2.95</td>
</tr>
<tr>
<td>MP_Sync</td>
<td>1372</td>
<td>0.472</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MP_Alltoall</td>
<td>1961</td>
<td>5.334</td>
<td>323681322.</td>
<td>119008.54</td>
</tr>
<tr>
<td>MP_IRecv</td>
<td>39600</td>
<td>0.100</td>
<td>14199.</td>
<td>5638.21</td>
</tr>
<tr>
<td>MP_WAIT</td>
<td>352330</td>
<td>5.593</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MP_Comm_split</td>
<td>48</td>
<td>0.054</td>
<td>14199.</td>
<td>5638.21</td>
</tr>
</tbody>
</table>

MESSAGE PASSING PERFORMANCE
## CP2K Timing Report

<table>
<thead>
<tr>
<th>SUBROUTINE</th>
<th>CALLS</th>
<th>ASD</th>
<th>MAXIMUM</th>
<th>AVERAGE</th>
<th>MAXIMUM</th>
<th>AVERAGE</th>
<th>MAXIMUM</th>
<th>AVERAGE</th>
<th>MAXIMUM</th>
<th>AVERAGE</th>
<th>MAXIMUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXIMUM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVERAGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SELF TIME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL TIME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- CP2K                         1 1.0  0.018  0.018  57.900  57.900
- qs_mol_dyn_low               1 2.0  0.007  0.008  57.725  57.737
- qs_forces                    11 3.9  0.262  0.278  57.492  57.493
- qs_energies_scf              11 4.9  0.005  0.006  55.828  55.836
- scf_env_do_scf               11 5.9  0.000  0.001  51.007  51.019
- scf_env_do_scf_inner_loop    99 6.5  0.003  0.007  43.388  43.389
- velocity_verlet             10 3.0  0.001  0.001  32.954  32.955
- qs_scf_loop_do_ot           99 7.5  0.000  0.000  29.807  29.918
- ot_scf_mini                  99 8.5  0.003  0.004  28.538  28.627
- cp_dbcsr_multiply_d          2338 11.6 0.005  0.006  25.588  25.936
- dbcsr_mm_cannon_multiply     2338 13.6 2.794  3.975  25.458  25.809
- cannon_multiply_low          2338 14.6 3.845  4.349  14.697  15.980
- ot_mini                      99 9.5  0.003  0.004  15.701  15.942
CP2K Timing Report

• Not just for developers!
  • Check that communication is < 50% of total runtime
  • Check where most time is being spent:
    • Sparse matrix multiplication - cp_dbcsr_multiply_d
    • Dense matrix algebra – cp_fm_syevd (&DIAGONALISATION), cp_fm_cholesky_ * (&OT), cp_fm_gemm
    • FFT – fft3d_*
    • Collocate / integrate – calculate_rho_elec, integrate_v_rspace

• Control level of granularity
  &GLOBAL
  &TIMINGS
    THRESHOLD 0.00001  Default is 0.02 (2%)
  &END TIMINGS
  &END GLOBAL
Summary

• First look for algorithmic gains
  • Cell size, SCF settings, preconditioner, choice of basis set, QM/MM, ADMM...
• Check scaling of your system
  • Run a few MD / GEO_OPT steps
  • Turn off outer SCF, keep inner SCF fixed
• Almost all performance-critical code is in libraries
  • Compiler optimisation –O3 is good enough
  • Intel vs gfortran vs Cray – difference is close to zero
• Before spending 1,000s of CPU hours, build lib[x]smm, libgrid, ELPA, FFTW3...
  • Or ask your local HPC support team 😊
Questions?