Jugene Case-Studies: Overview

- Case Study: PEPC
- Case Study: racoon
- Case Study: QCD
Case Study: PEPC

- Parallel tree code PEPC: Pretty Efficient Parallel Coulomb-solver
- ‘Hashed-oct-tree’ algorithm using multipole expansions
- Applications:
  - petawatt laser-plasma acceleration
  - strongly coupled Coulomb systems
  - stellar accretion discs
  - vortex-fluid simulation
- Future version:
  - magnetic fields, implicit integration scheme

Paul Gibbon,
Jülich Supercomputing Center, Forschungszentrum Jülich
http://www.fz-juelich.de/jsc/pepc
PEPC: Application Laser-produced proton beams

- Hot electron cloud $T \sim \text{MeV}$
- Electric field $\sim 10^{12} \text{Vm}^{-1}$
- PW Laser: 100 J/100 fs
- Solid target (Al, Au foil)
Case Study: PEPC, main steps

1. Domain decomposition

2. Local trees

3. Non-local trees + interaction lists
Case Study: PEPC, 2D/3D domain decomposition

100 particles
4 procs

200 particles
8 procs
Case Study: PEPC, interaction lists

Multipole acceptance criterion (MAC): $s/d < \theta$
Case Study: PEPC, Fetching non-local multipole terms

Need children of node j from $P_2$

Ship children of node j back to $P_0$

$P_0$, $P_1$, $P_2$
Case Study: PEPC, Parallel Scalability
PEPC: PetaScale analysis

- good scaling up to 8192 core on Blue Gene/P
- bottlenecks:
  - internal data structures
  - data exchange with MPI Alltoallv
  - meta-information needed to exchange data between tasks
- possible solutions:
  - other internal data structure
  - hybrid parallelization
  - larger problem sizes to increase computation / communication ratio
racoon: Overview

- refined adaptive computations with object-oriented numerics
- software framework for time dependent PDE (HD, MHD)
- uses an adaptive grid, with a octree block structure
- main scientific focus: current sheets and magnetic reconnection, turbulence
- written in C++, utilizes MPI

Jürgen Dreher, Computational Physics, Ruhr-Universität Bochum
http://www.tp1.ruhr-uni-bochum.de/ Forschung racoon

current density of a magnetic flux tubes (sun surface)
raccoon: Parallelization

- parallelization realized by distributing the blocks
- each block needs to communicate to its neighbors → reduce the comm. by keeping neighbors on the same 'compute node'
- the hilbert-curve maps all blocks (in 2D/3D/nD) to a 1D curve, preserving neighboring properties → load balancing
**racoon: sample scaling problem**

*a simple example how a vector variable prevents scaling*

- in *racoon* all MPI proc. compute the meta info (grid structure, block distribution, communication partners) themselves
- example: one vector contains all communication partners, approx. 16 byte per pair
- its size is \( n^2 \), where \( n \) is the size of MPI tasks -> NONLINEAR size
- vector's size in numbers:
  - small cluster (64 CPUs) \(~ 16\text{kB}\)
  - JUMP (1000 CPUs) \(~ 4\ \text{MB}\)
  - JUGENE (16000 CPUs) \(~ 800\ \text{MB}\)
- this vector would use more than 512 MB RAM -> will not run on JUGENE, even when the physical -numerical- problem size is appropriate
- conclusion: meta data, which scales (nonlinear) with \( n \), might become a problem
  → distribute meta information
racoon: Scaling

![Graph showing scaling of racoon with different processor counts and block sizes.](image-url)
Lattice QCD: Overview

Lattice QCD (LQCD) is defined on a 4 dim. periodic lattice; LQCD is a way to define QCD in a mathematically precise way.

Key ingredients are:

- The **Quarks** living on the lattice sites
- The **Gluons** living on the lattice links
- Typically the LQCD **action** connects only neighboring sites (plain Wilson)

*Simulations of LQCD are the only available method to directly access the low energy regime of QCD*

Key parts of simulation:

- Hybrid Monte Carlo (HMC),
- Inversion of SU3 Wilson matrix

Stefan Krieg,
Jülich Supercomputing Center, Forschungszentrum Jülich
FB. C. Mathematik und Naturwissenschaften, Bergische Universität Wuppertal
QCD particle spectrum

Preliminary
BG/P special features used by LQCD: torus network

**Wilson kernel communication pattern**

- match 4 dim. period physics lattice to BG/P torus network
- put 3 dimensions along torus directions
- use local SMP memory based MPI communication for 4th dim.

Core0 → Core1
↑
Core2 ← Core3

→ 4 dimensional torus
BG/P special features used by LQCD: DMA controller
BG/P special features used by LQCD: DMA controller

DMA is capable of:

- **Direct-put**: put data into memory of destination node (used by LCQD)
- **MemFifo comms**: put data into reception fifo on destination node
- **Remote-get**: put a descriptor into injection fifo on destination node
- **Prefetch-only**: prefetch data into L# (no transfer)
- Destination node can be the node itself (local transfer)
- FIFO contains message descriptors

→ DMA is “directly” programmable by SPI
LQCD: Overlapping Calculation and communication

- Wilson kernel is a sparse matrix vector multiplication
- Sparse: memory footprint scales linearly with N
- DMA controlled data exchange with direct neighbors in background
  → full overlap of communication and calculation
BG/P special features used by LQCD: “Double Hummer” FPU

Instructions are optimized for complex arithmetic:
- 32 primary + 32 secondary registers
- Capability to load 16 Byte quadwords
- 5 stages pipeline
- only two instructions required for complex multiplication

\[ A \times B = C \rightarrow Re(C) = (Re(A)Re(B) - Im(A)Im(B)) \]
\[ Im(C) = (Im(A)Re(B) + Re(A)Im(B)) \]

Instruction 1: cross copy primary multiply
Instruction 2: cross mixed negative second multiply-add
**BG/P special features used by LQCD: Intrinsics & Assembly**

**Intrinsics** (built-in functions)
- provided by IBM XL compilers
- map to (e.g. floating point) assembly instructions, (e.g. \_lfpd, \_stfpd, \_fpmadd and \_dcbt)
- Intrinsics operate on “double _Complex” variables that map to registers
  - Comparatively easy to use
  - LQCD code has large parts optimize using intrinsics
  - The compiler has great influence on the performance, if using intrinsics

**Assembly instructions**
- For more control use (gcc inline) assembly
- Kernel serial code written in assembly
- Uses explicit prefetches
- All scheduling and register allocation done by hand
  - Performance typically another 10% better compared to intrinsics
  - Code generation typically 10 times slower compared to intrinsics
LQCD: Results

Wilson kernel shows
- almost perfect strong scaling
- large scale range
- perfect weak scaling
- and reaches 37% of absolute peak

Full talk:  Journée Blue Gene/P
IDRIS(CNRS), 08.04.08
Blue Gene/P: A Optimization Strategy

• Single Core Performance
  – Compiler options
  – SIMD, using “Double Hummer”

• Using Libraries, whenever possible (ESSL)

• Scaling Problems
  – Storing global meta information locally can fulfill the memory $O(N)$
  – (Nested) Loops over number of MPI tasks can be time consuming if running on 16k tasks

• Use special features of Blue Gene/P
  – different network (torus, tree, …)
  – Overlapping Communication and Computation
  – “Double Hummer”, Intrinsics

• Change algorithm / data structures if other (not so effective) scales better on large number of tasks

• To get the last % → Assembly and SPI low level programming