Physics Performance and Operating System Overview of QCDOC

Peter Boyle

University of Edinburgh
Columbia University

- QCD Simulation Code
- Node-kernels
- Host Software
- Partitioning

QCD Simulation Code

- Perform QCD path integral by Monte-Carlo .
- Cost dominated by solution of quark Dirac equation to incorporate sea quark effects.
- ECFA: *each* sample $\simeq 10^{20} \times \left(\frac{M_\pi}{M_{PS}}\right)^{2.7}$ flops.
- Many many Teraflop years for physical sea quarks $\Rightarrow$ Parallelise.
- Domain decomposition on mesh of N processors.
- Dirac matrix $\Rightarrow$ nearest neighbour interaction. Sparse matrix performance depends on $V_{local} = \frac{V_{global}}{N}$. QCDOC designed for V_{local} down to to 2^4.
- Iterative solvers $\Rightarrow$ global summation $\propto N^{\frac{1}{4}}$.

Machine balance chosen to maximise scalability for QCD.

- Good single node performance 1 Gflop peak. Edram allows 50% for double precision QCD.
- 550ns latency, $24 \times$ 50MB/s to neighbours. Communication and computation balanced for Wilson quarks with $V_{local} = 2^4$.
- Fast global summation $\simeq 10\mu$s.

Single Node FP Kernels

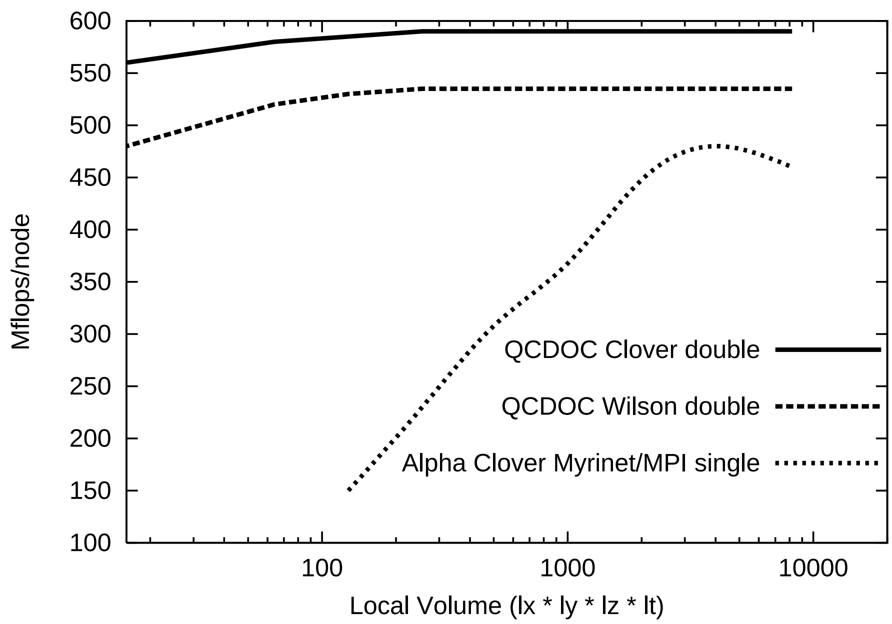
Generate QCD assembler kernels using a C++ package.
Targets Alpha, Power, Ultrasparc.
500MHz target frequency.

Operation	Mflops/node
SU3-SU3	800
SU3-2spinor	780
DAXPY	190
ZAXPY	450
DAXPY-Norm	350
CloverTerm/asm	790
CloverTerm/gcc	150
CloverTerm/xlc	300

Multi-node sparse matrix performance

Operation	Local Vol.	Mflops/node
Wilson D_{eo}	2^4	470
Wilson D_{eo}	4^4	535
Clover D_{eo}	2^4	560
Clover D_{eo}	4^4	590
Staggered D_{eo}	2^4	370
Staggered D_{eo}	$2^2.4^2$	430

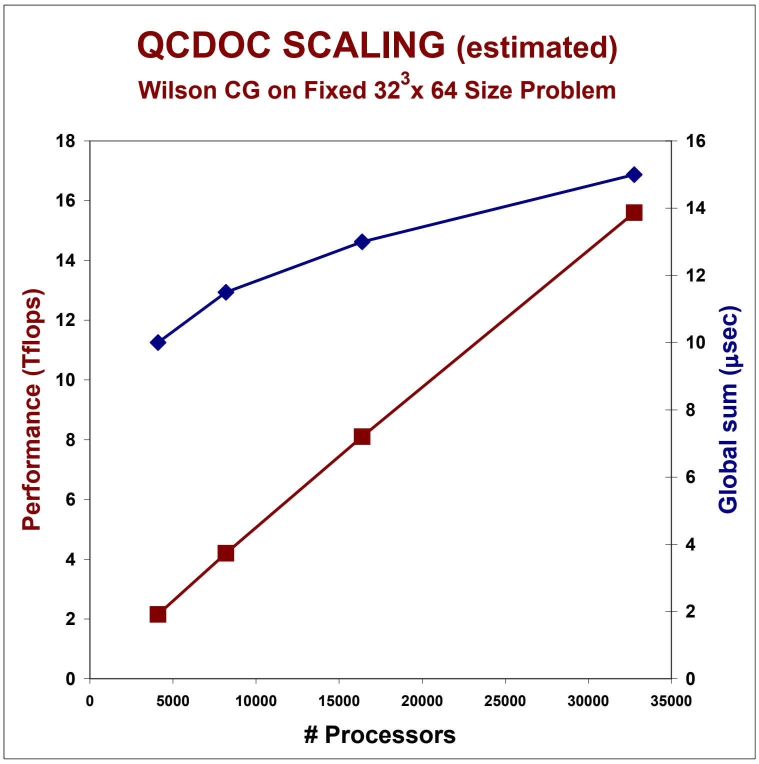
Wilson D_{eo} takes around 22μs on $V_{local} = 2^4$.
16 distinct communications
Send + Receive, backwards and forwards, 4 dimensions.
Latency of 550ns is amortized across 8 transfers
Overlap communication with computation.



Estimated Scalability

Based on cycle accurate simulation.
Wilson CG performance on a $32^3 \times 64$ Lattice.
Clover and Domain Wall Fermions even more scalable.

Nodes	$M^\dagger M$	Gsum	Sust. Tflops
4096	2620μs	10 μs	2.15
8192	1310μs	11.5μs	4.2
16384	680 μs	13 μs	8.1
32768	340 μs	15 μs	15.6

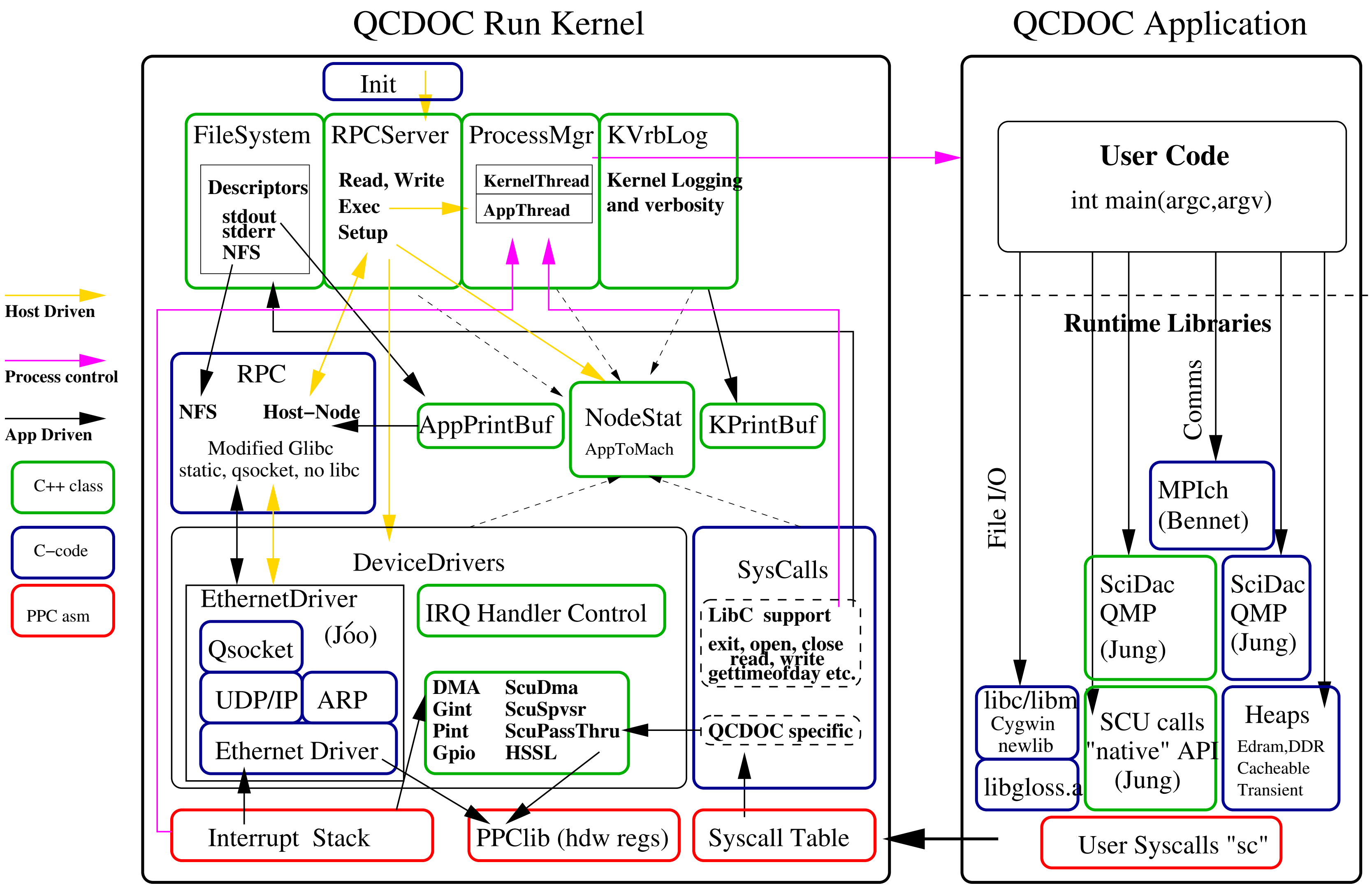


Operating System

- Boot machine
- Diagnose hardware errors
- Run one program on each CPU
- Service Comms and I/O requests

Robust, Simple, Efficient
Debug hardware with software!

- No-nonsense Unix SMP front end.
- Controlled boot sequence + lean kernel
- Run one process and run it well.
  VERY tightly coupled - no scheduling.
- Memory protection but not translation
  Zero copy DMA + Never miss TLB.


QCDOC Run Kernel
QCDOC Application
Init
FileSystem
Descriptors
stdout
stderr
NFS
RPCServer
Read, Write
Exec
Setup
ProcessMgr
KernelThread
AppThread
KVrbLog
Kernel Logging
and verbosity
User Code
int main(argc,argv)
Runtime Libraries
Comms
File I/O
RPC
NFS
Host–Node
Modified Glibc
static, qsocket, no libc
AppPrintBuf
NodeStat
AppToMach
KPrintBuf
MPIch
(Bennet)
DeviceDrivers
EthernetDriver
(Jóo)
Qsocket
UDP/IP
ARP
Ethernet Driver
IRQ Handler Control
DMA
Gint
Pint
Gpio
ScuDma
ScuSpvsr
ScuPassThru
HSSL
SysCalls
LibC support
exit, open, close
read, write
gettimeofday etc.
QCDOC specific
SciDac
QMP
(Jung)
SciDac
QMP
(Jung)
libc/libm
Cygwin
newlib
libgloss.a
SCU calls
"native" API
(Jung)
Heaps
Edram,DDR
Cacheable
Transient
Interrupt Stack
PPClib (hdw regs)
Syscall Table
User Syscalls "sc"
Host Driven
Process control
App Driven
C++ class
C–code
PPC asm

Host-O/S: Design rationale

Should boot, partition and run machine

- **Dynamic** software partitioning
  $\Rightarrow$ node ownership changes with partitioning
  Shared memory desirable.

- High bandwidth requirements
  parallelise over multiple Gbit cards on host.

- Non-blocking reliable UDP communication.

- POSIX threads library allows all of the above.

Multi-threading

Encapsulate MT properties in template and base classes.

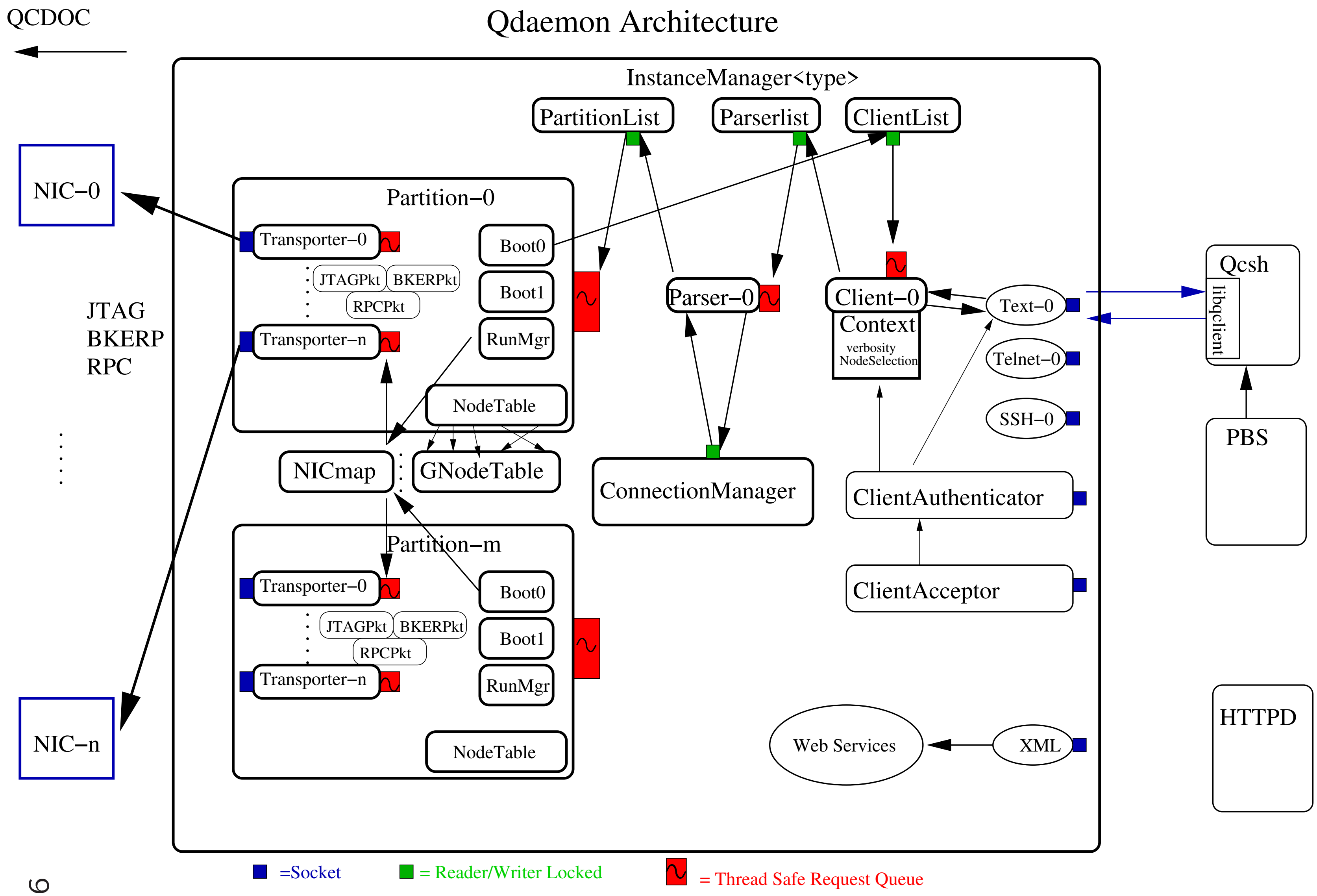
- Locking (mutex), Waiting (condition var), RWLocking (reader-writer locks), Runnable (thread creation)

- Who deleted my pointer!
  InstanceManager(thread safe object list)

- Who stomped on my data!
  SafeQueuedCall (thread safe request queue)

- Who will retransmit my packet!
  Transporter (*Reliable* async-I/O interface)

Qdaemon Architecture



Partitions

- Machine $2k \times 2l \times 2m \times 2 \times 2 \times 2$ hyper-torus.
- Partitions are 6d hyper-cuboid subvolumes.
- Application sees 1,2,3,4,5 or 6 dim Torus

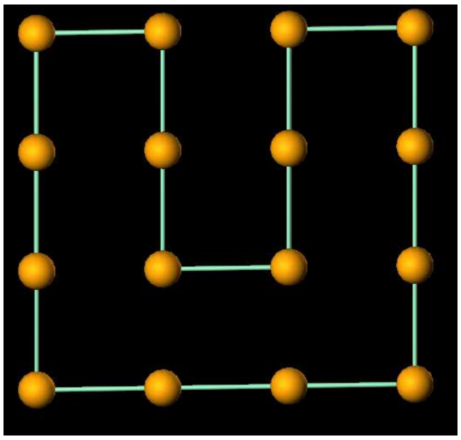
Root Partition can create and destroy subpartitions
User Partition can remap topology.

- Machine axes mutually orthogonal
  and translationally invariant!
- Lower-left, upper-right 6-tuples specify cuboidal partition.
- Subset of machine axes may be periodic in partition.
- Locally redefine Application to Machine axis map
  Can change dimensionality.

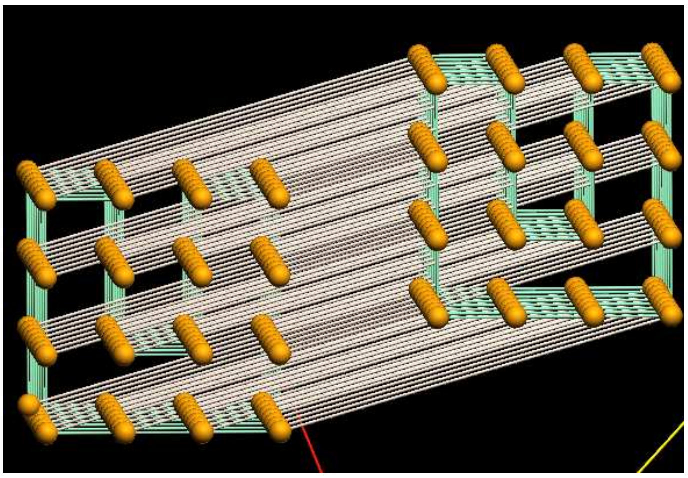
Dimension Folding

Simple case:
Fold two machine axes m_1, m_2 lengths l_1, l_2
Forms one application axis length $l_1 \times l_2$
Requirement: one of l_1 or l_2 is even.

$l_1 \cdot l_2 \cdot l_3 \cdot l_4 \cdot l_5 \cdot l_6 \to (l_1 \times l_2) \cdot l_3 \cdot l_4 \cdot l_5 \cdot l_6 \cdot 1$



$4 \cdot 4 \longrightarrow 16$



$4 \cdot 4 \cdot 8 \cdot 2 \longrightarrow 16 \cdot 8 \cdot 2$

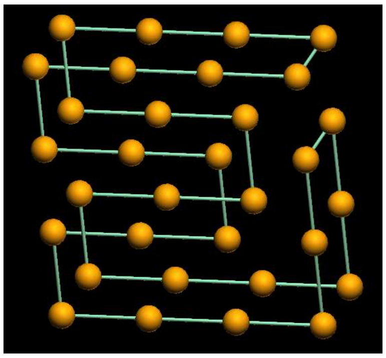
Remapped partition is a properly connected torus

- Translational invariance the in 3,4,5,6 directions guarantees the 2-d $l_1 \times l_2$ curve is replicated.

- Orthogonality guarantees the links exist to make up remapped cartesian grid.

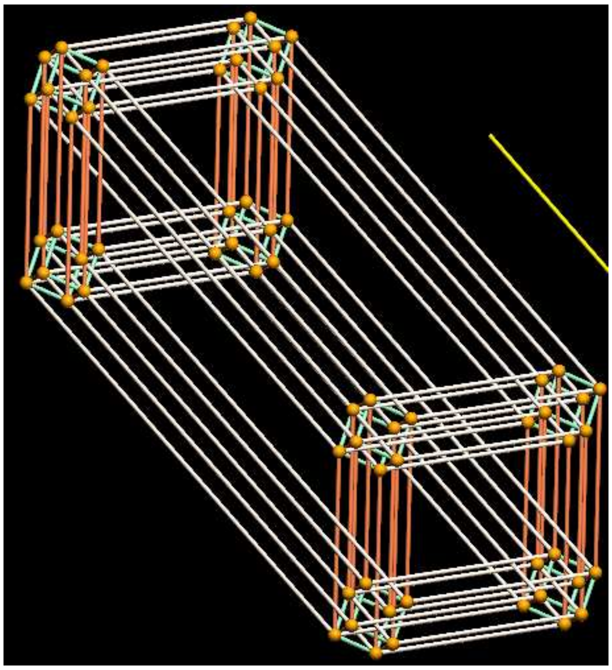
Keep on folding!

m_3 Orthogonal to both m_1 and m_2 $\Rightarrow$ iterate.
$(l_1 \times l_2 \times l_3) \cdot l_4 \cdot l_5 \cdot l_6 \cdot 1 \cdot 1$



$4 \cdot 4 \cdot 2 \longrightarrow 32$

m_4 and m_5 orthogonal to m_1 and m_2
$(l_1 \times l_2) \cdot (l_3 \times l_4) \cdot (l_5 \times l_6) \cdot 1 \cdot 1 \cdot 1$



2^6 motherboard $\longrightarrow 4 \cdot 4 \cdot 4$

Summary

Physics code achieves high fractions of peak and outstanding scalability.

Simple node kernel design avoids many of the pitfalls MPP's based on Unix like nodes.

Controlled software environment allows for

- Debug and instrumentation of hardware.

- Low software overhead on communication.

Use Open Source where possible for standard interfaces.

Operating system supports variable dimensionality...

Good for all sparse matrix problems on a Cartesian grid, and particularly good for QCD.