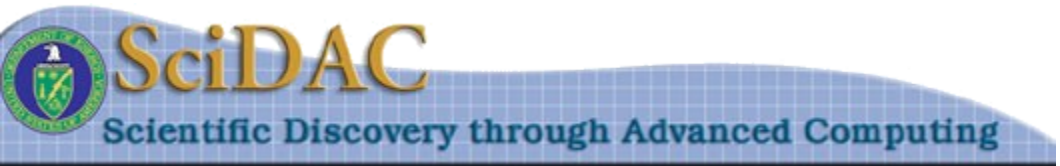
Accelerating past the petascale

Robert J. Harrison
harrisonrj@ornl.gov
robert.harrison@utk.edu



Mission of the ORNL National Leadership Computing Facility (NLCF)

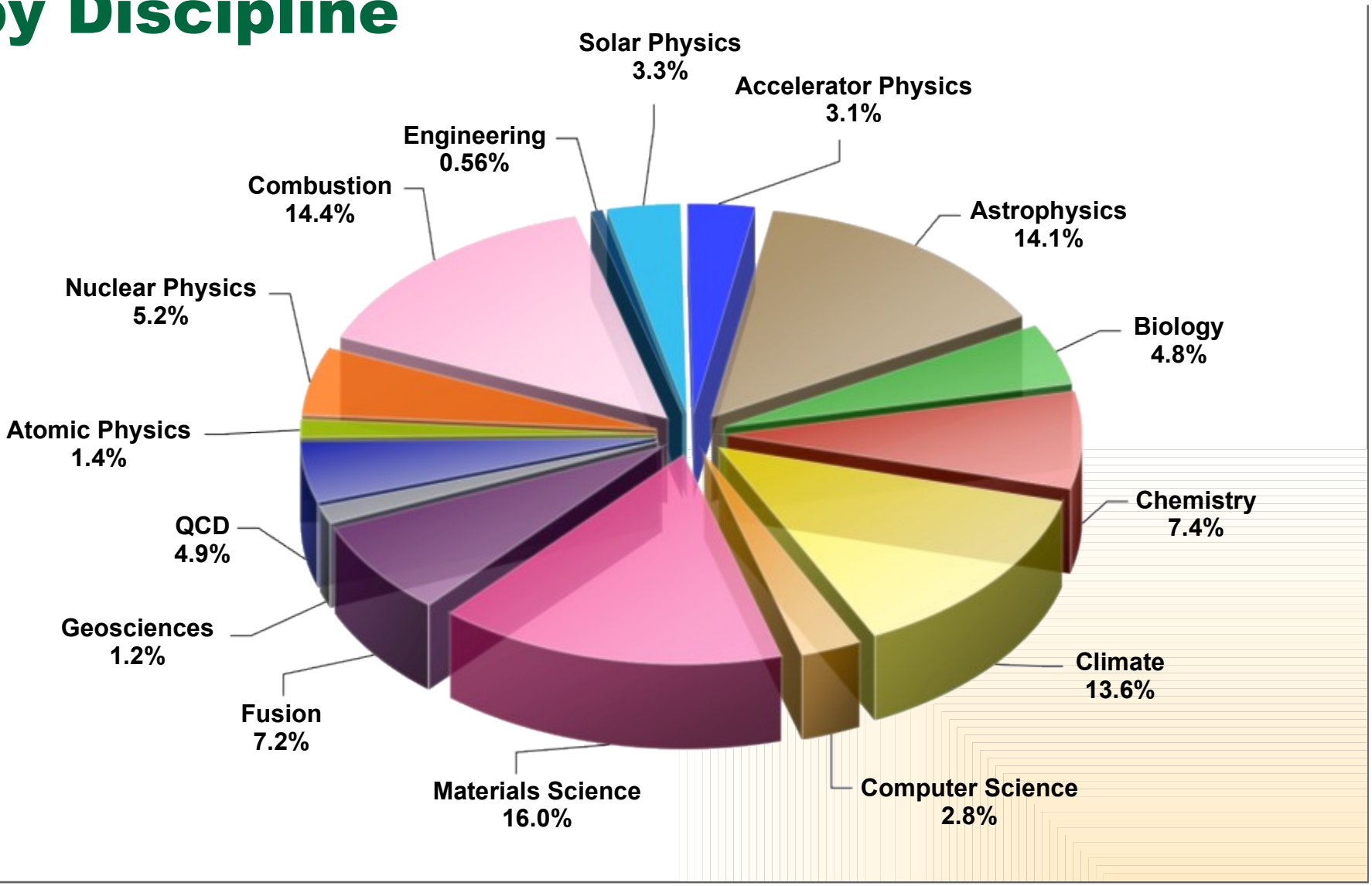
- field the most powerful capability computers for scientific research
- select a few time sensitive problems of national importance that can take advantage of these systems
- join forces with the selected scientific teams to deliver breakthrough science.

Cray XT5 at ORNL > 1 Pflop/s in November 2008

Jaguar	Total	XT5	XT4
Peak Performance	1,645	1,382	263
AMD Opteron Cores	**181,504**	**150,176**	**31,328**
System Memory (TB)	**362**	**300**	**62**
Disk Bandwidth (GB/s)	**284**	**240**	**44**
Disk Space (TB)	**10,750**	**10,000**	**750**
Interconnect Bandwidth (TB/s)	**532**	**374**	**157**

The systems will be combined after acceptance of the new XT5 upgrade. Each system will be linked to the file system through 4x-DDR Infiniband

ORNL INCITE 2008 Allocations by Discipline



INCITE: Innovative and Novel Computational Impact on Theory and Experiment

Univ. of Tennessee & ORNL Partnership
National Institute for Computational Sciences

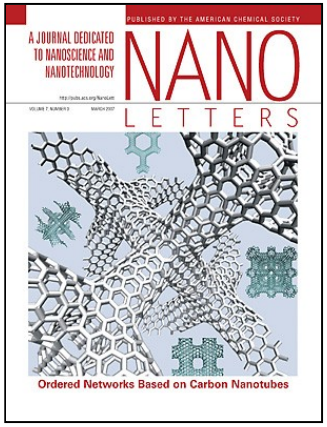
- UT is building a new NSF supercomputer center from the ground up
  - Building on strengths of UT and ORNL
  - Operational in May 2008
- Series of computers culminating in a 1 PF system in 2009
  - Initial delivery (May 2008)
    - 4512 quad-core Opteron processors (170 TF)
  - Cray "Baker" (2009)
    - Multi-core Opteron processors; 100 TB; 2.3 PB of disk space

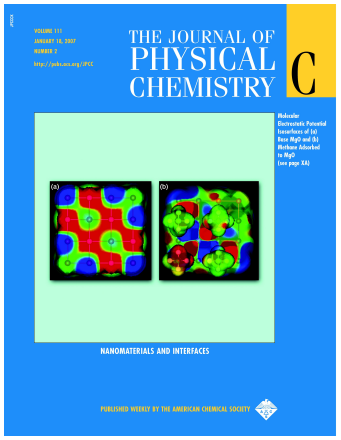
Leadership computing in chemistry

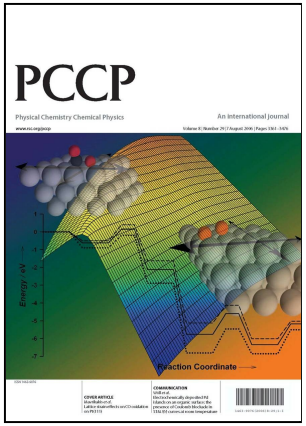
- **Definitive, benchmark computations**
  - **The scale and fidelity of petascale simulation will answer truly hard questions about real systems. Fully quantitative computations are central to fundamental understanding and to enabling rational design.**
- **Integration of experiment and theory**
  - **Fast turnaround of reliable simulations is already enabling the intimate integration of theory and simulation into chemistry, which is a predominantly experimental discipline.**

An Integrated Approach to the Rational Design of Chemical Catalysts NCCS Incite Project

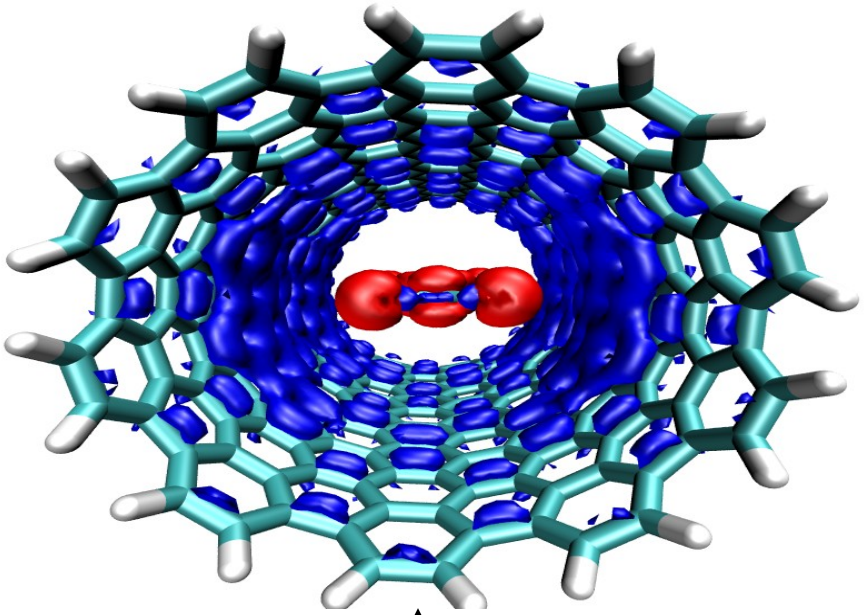
Robert J. Harrison: PI	**Oak Ridge National Laboratory, University of Tennessee**
Edoardo Aprà	**Oak Ridge National Laboratory**
Jerzy Bernholc	**North Carolina State University**
A.C. Buchanan III	**Oak Ridge National Laboratory**
Marco Buongiorno Nardelli	**North Carolina State University**
James M. Caruthers	**Purdue University**
W. Nicholas Delgass	**Purdue University**
David A. Dixon	**University of Alabama**
Sharon Hammes-Schiffer	**Pennsylvania State University**
Duane D. Johnson	**University of Illinois at Urbana Champaign**
Manos Mavrikakis	**University of Wisconsin at Madison**
Vincent Meunier	**Oak Ridge National Laboratory**
Mathew Neurock	**University of Virginia**
Steven H. Overbury	**Oak Ridge National Laboratory**
William F. Schneider	**University of Notre Dame**
William A. Shelton	**Oak Ridge National Laboratory**
David Sherrill	**Georgia Institute of Technology**
Bobby G. Sumpter	**Oak Ridge National Laboratory**
Kendall T. Thomson	**Purdue University**
Roberto Ansaloni	**Cray**
Carlo Cavazzoni	**CINECA, Italy**

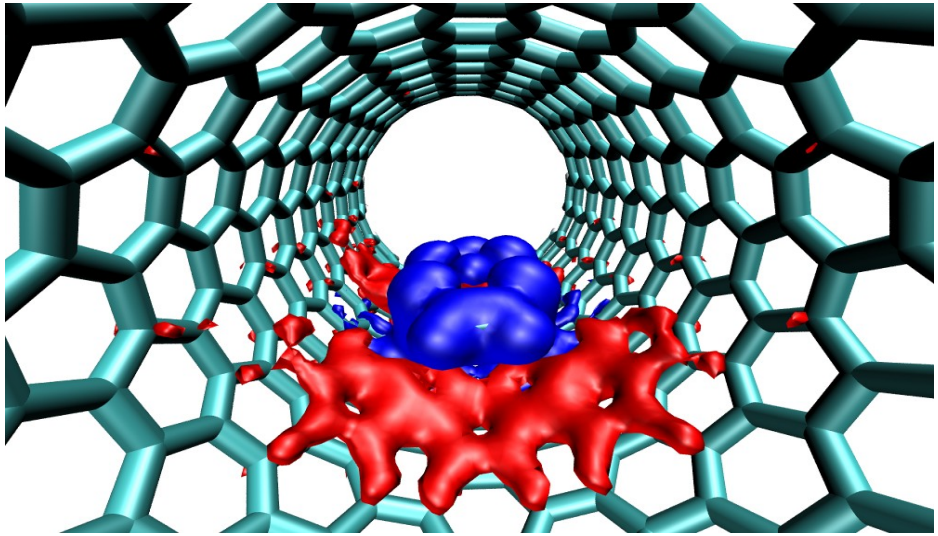




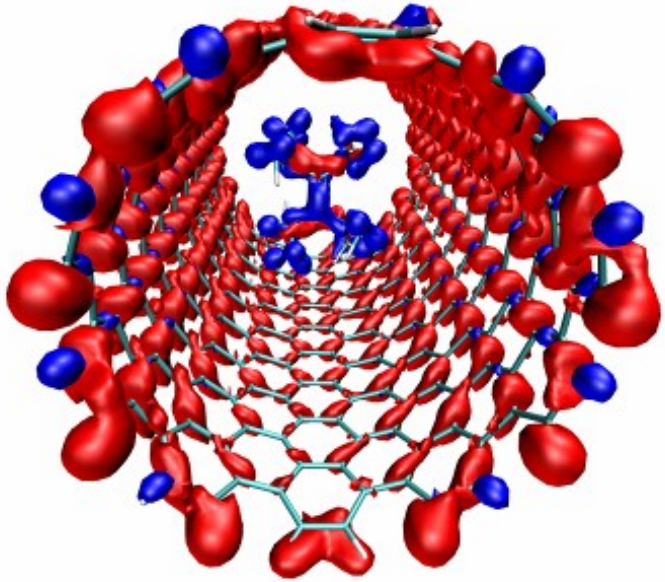


Amphoteric Doping of Carbon Nanotubes by Encapsulation of Organic Molecules: Electronic Transport and Quantum Conductance





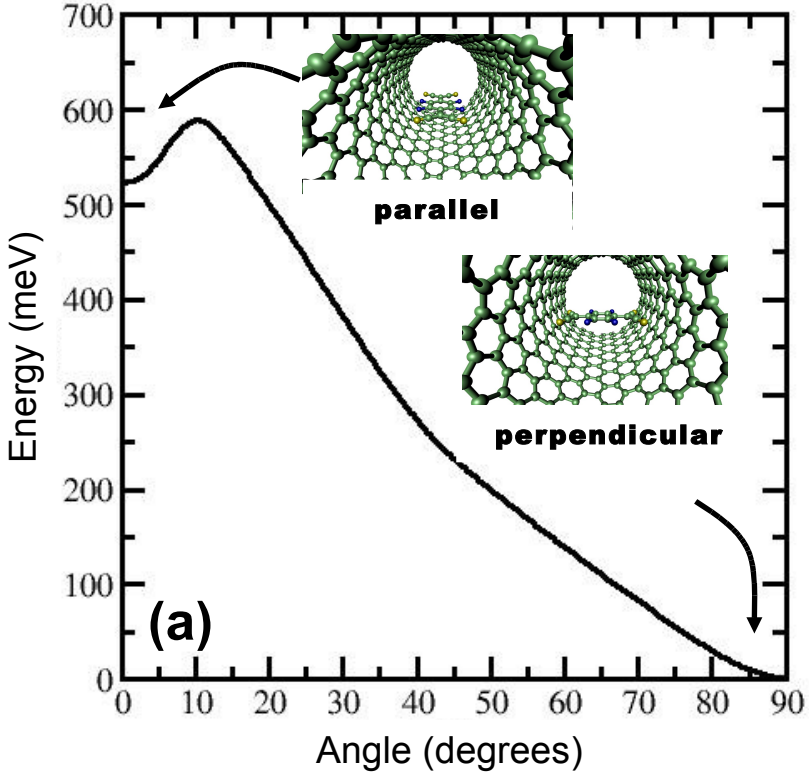
p-doped tube : holes are transferred from F_4-TCNQ to the nanotube

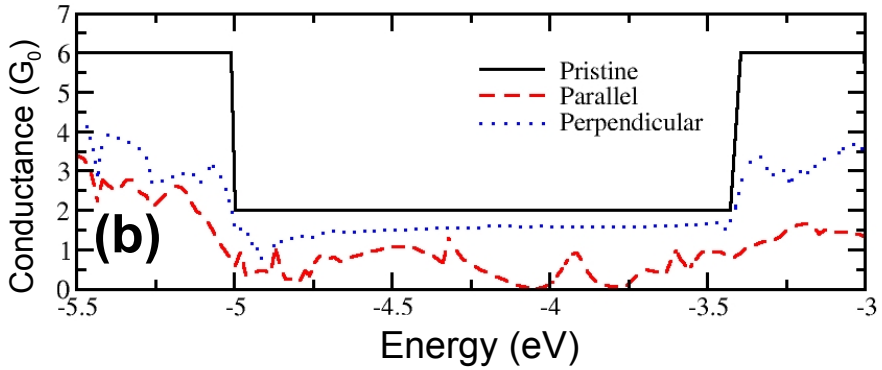


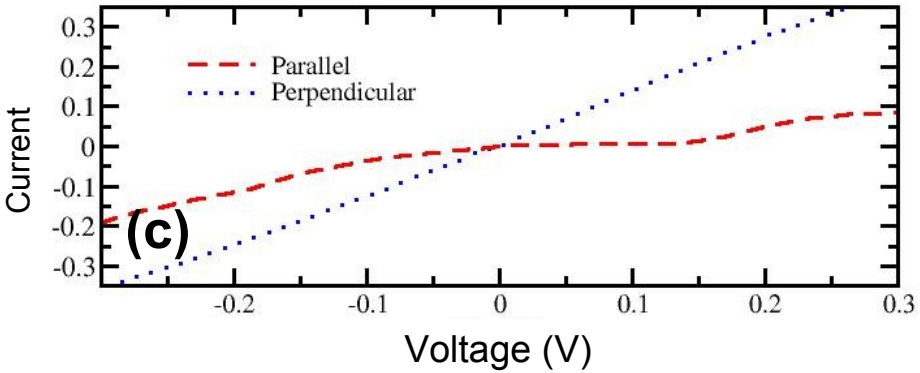
n-doped tube : holes are transferred from the tube to TTF and TDAE

Meunier, Sumpter

F4TCNQ: structure and transport



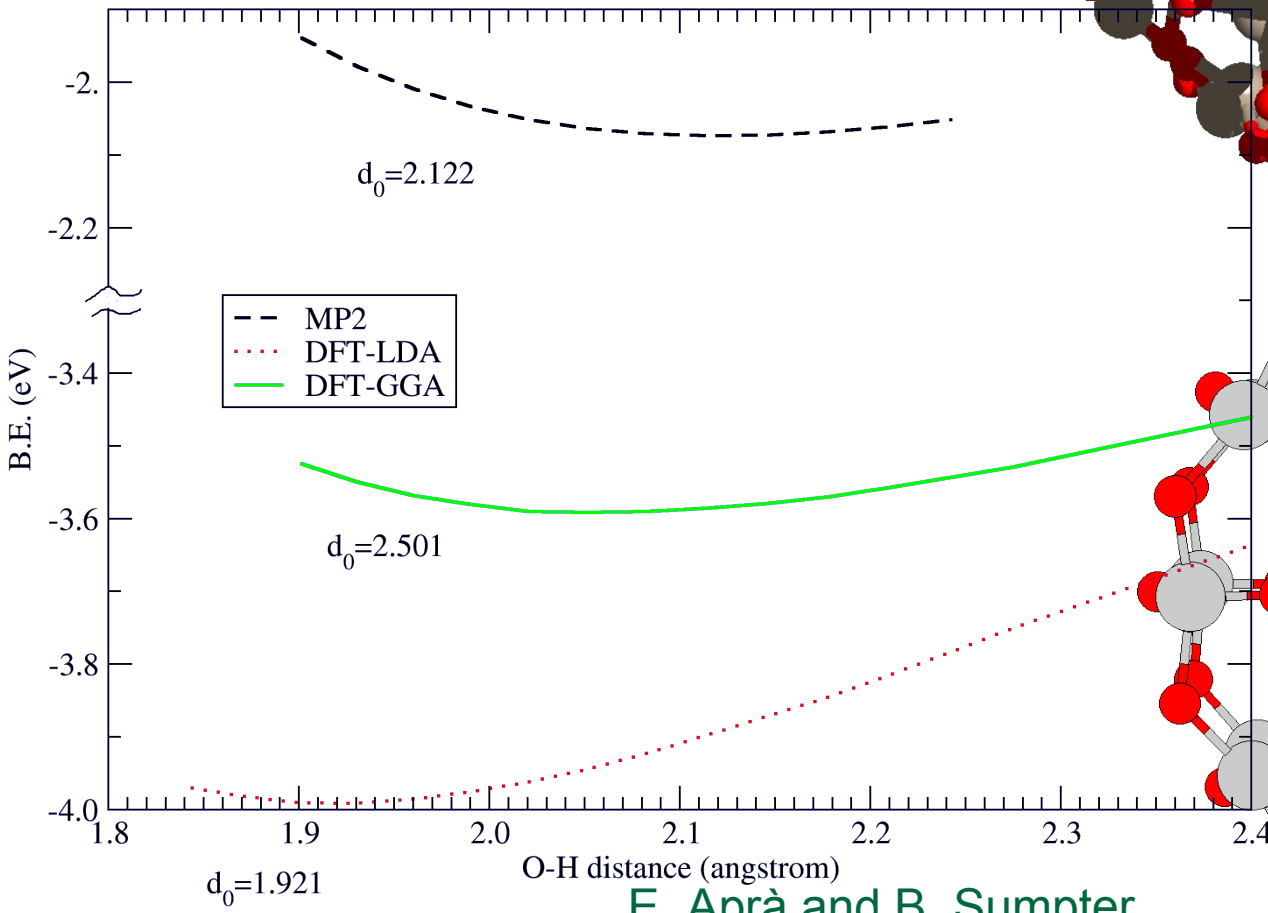


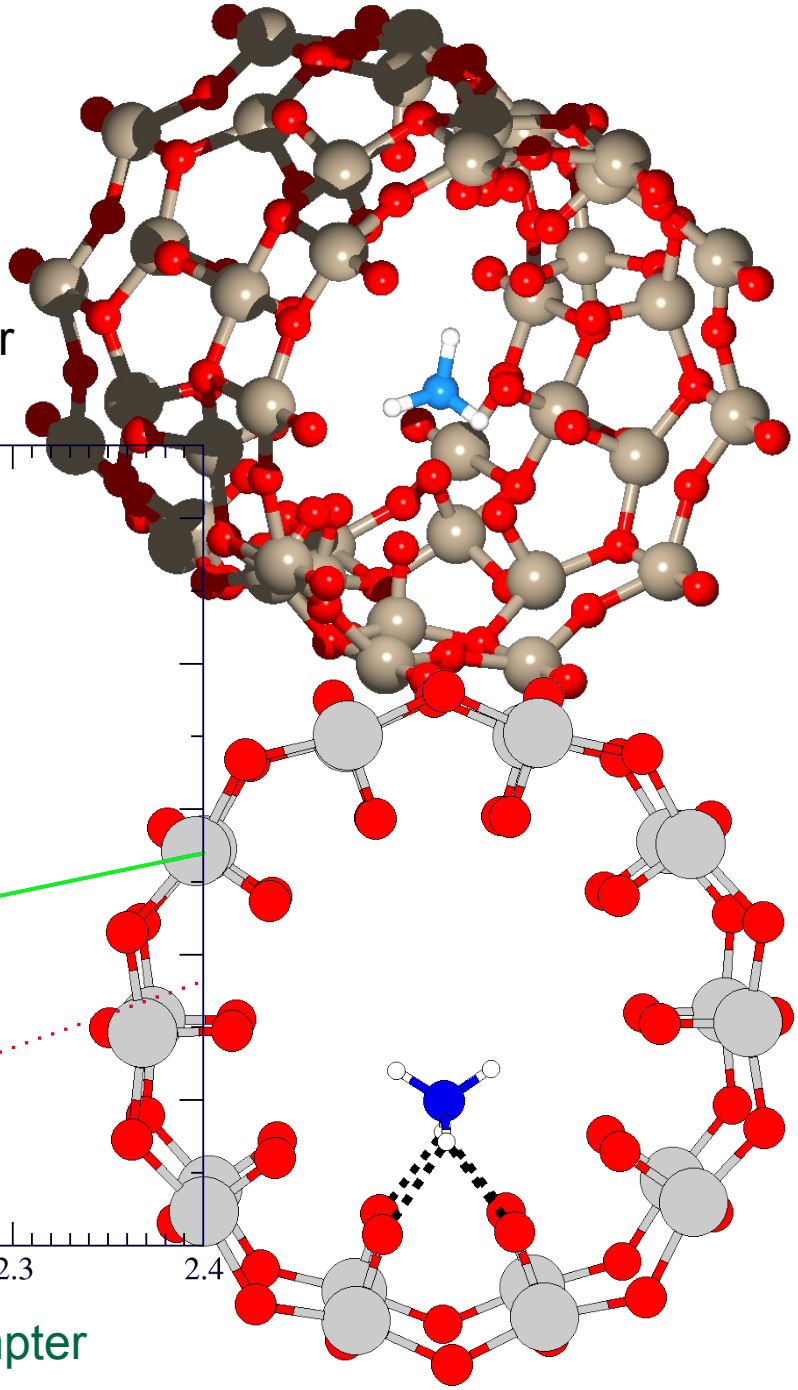


Kalinin, Meunier, Sumpter

V_2O_5 nanotubes

- Applications in energy storage (capacitor)
- V2O5, Nb2O5, and Ta2O5 armchair and zigzag and for different diameters
- Will transition metal oxide tubes maintain tubular structure when N is substituted for O?



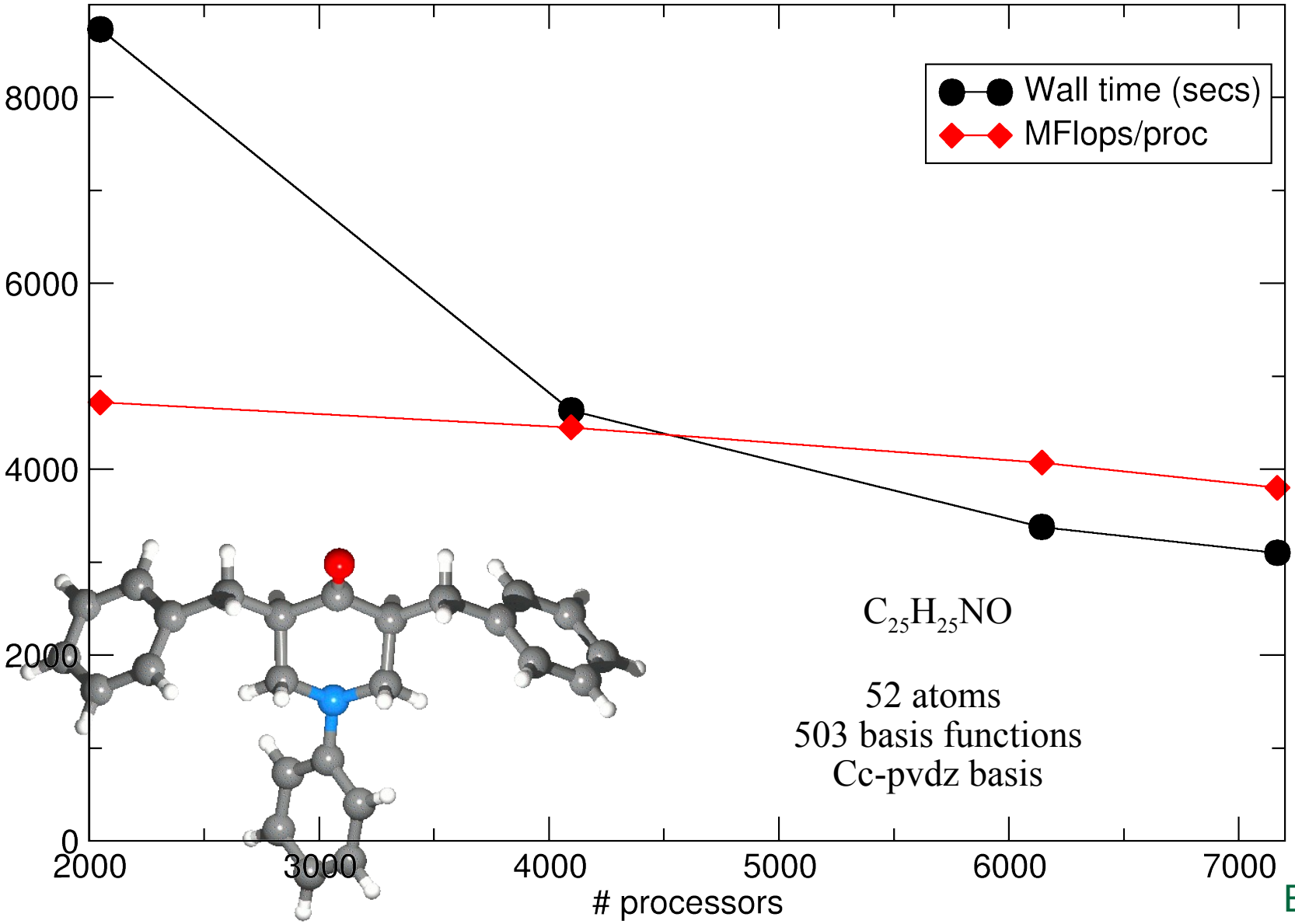


E. Aprà and B. Sumpter

NWChem port for the Cray-XT
E. Aprà & V.Tipparaju, ORNL

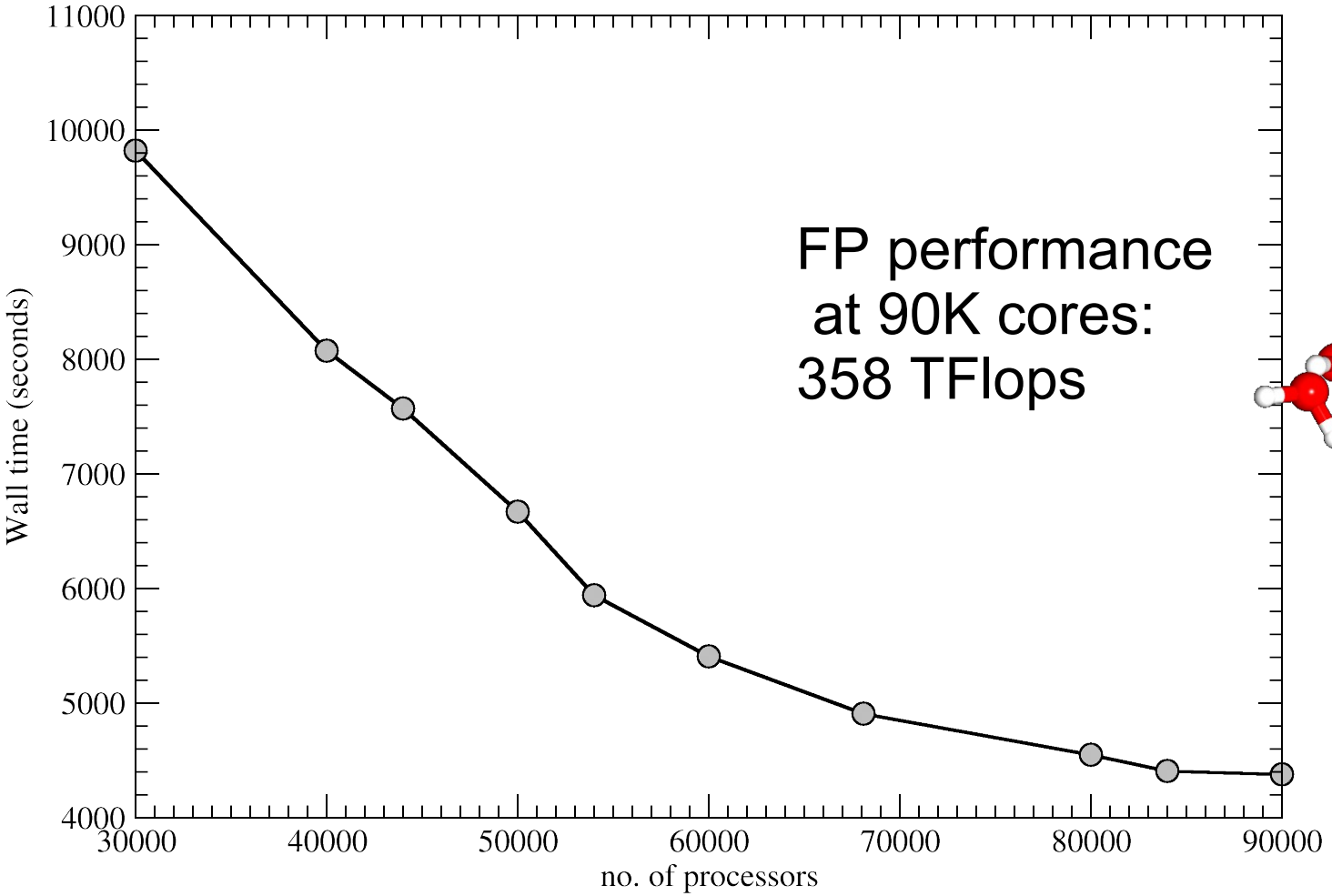
- Custom ARMCI port uses the Portals library for inter-node communication
- SMP aware: uses SysV shared memory for intra-node communication
- Server-layer for calls that do not map directly onto the network

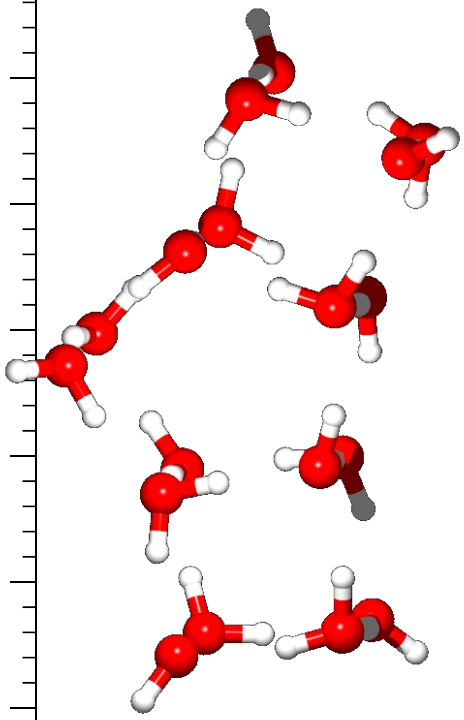
CCSD(T) run on Cray XT4



E. Aprà

CCSD(T) run on Cray XT5 : 18 water



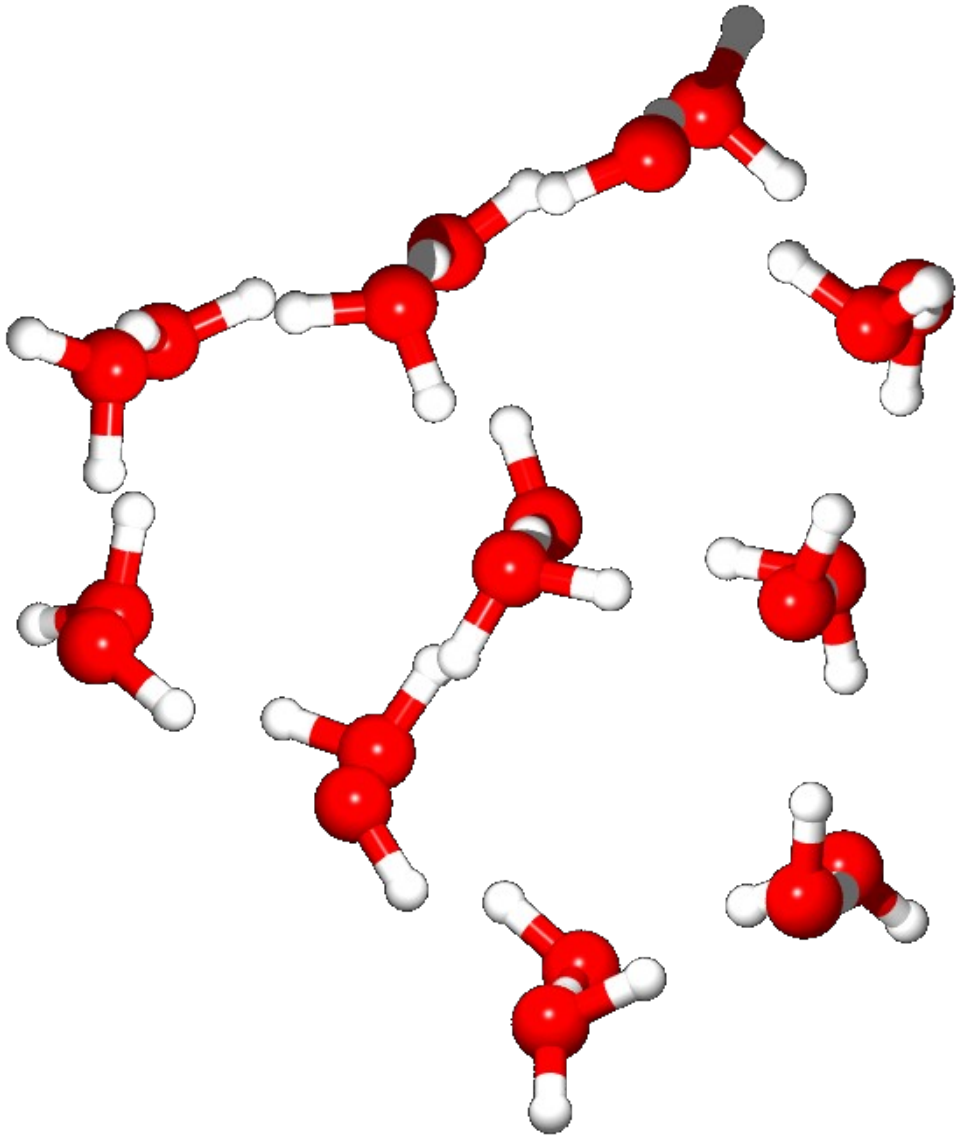


$(H_2O)_{18}$

54 atoms
918 basis functions
Cc-pvtz(-f) basis

E. Aprà

CCSD(T) run on Cray XT5 : 20 water



Floating-Point performance
at 92K cores:
475 TFlops
Efficiency > 50% of peak

$(H_2O)_{20}$

60 atoms
1020 basis functions
Cc-pvtz(-f) basis

E. Aprà

Next generation ORNL NLCF

- ORNL has proposed a system to meet DOE's requirement for 20-40 PF of compute capability split between the Oak Ridge and Argonne LCF centers
- ORNL's proposed system will be based on accelerator technology
  - includes software development environment
- We plan to deploy the system in late 2011 with users getting access in 2012
- Watch this space for more details soon

http://www.nccs.gov/

Why accelerators and why now?

- **Path to exascale**
  - **power 0.1 → 100 GFLOP/Watt**
  - **memory 0.3 → 0.03 byte/FLOP**
  - **cores 8 → 64-1024+ per node**
  - **number of cores 100K → 100M**
  - **concurrency $10^6 \rightarrow 10^9$**
- **Must express & exploit parallelism at all levels**
  - **Currently only have coarse (MPI) and medium (within SMP) granularity**
- **Mainstream CPUs no longer provide sufficient density of computation – convergence when/if?**

O(1) programmers
O(10,000) nodes
O(100,000) processors
O(100,000,000) threads

- Complexity kills … sequential or parallel
- Expressing/managing concurrency at the petascale
  - It is too trite to say that the parallelism is in the physics
  - Must express and discover parallelism at more levels
  - Low level tools (MPI, Co-Array Fortran, UPC, …) don't discover parallelism, hide complexity, ease abstraction
- Management of the memory hierarchy
  - Memory will be deeper ; less vendor uniformity
  - Need tools to automate and manage, even at runtime

GPGPUs v.s. CPUs

- How to rationalize the extreme acceleration seen in published “benchmarks” comparing GPUs to x86 CPUs
  - 100x, 200x, 400x, ... !!!???
  - What is attributable to hardware?
  - What is due to compilers/libraries?
  - What is the effort involved?
  - What should we be promising users of our S/W?

Concrete example hardware

- **Intel Core i7 920 @ 2.67 GHz, 1.33 GHz DDR3**
  - **Quad core**
    - **Each core 4 double and 8 single precision FLOP/cycle**
    - **25 (35) GB/s memory bandwidth**
    - **32KB L1/core data, 256KB L2/core, 8M L3/shared**
- **NVIDIA Tesla C1060 @ 1.3 GHz**
  - **240 cores**
    - **Each core 1 single precision MADD + 1 MUL / cycle and 1/8 double precision MADD / cycle**
    - **102 GB/s memory bandwidth**
    - **64KB registers + 16KB shmem + 8KB texture cache + 64KB constant cache per MP (i.e., per 8 cores)**

Concrete example hardware

- **Speed Ratios** — **Tesla : Intel 920**
  - Ratio of single precision FLOP/s — 11 : 1
  - Ratio of double precision FLOP/s — 1.8 : 1
  - Memory bandwidth — 4 : 1
  - On-chip memory — 0.51 : 1
- **Latencies and latency hiding mechanisms**
  - Must consider in general
  - Should not be relevant for this particular benchmark (more to this story follows)

Concrete example algorithm

- Metropolis Monte Carlo
  - General and powerful algorithm for multi-dimension integration
  - We abuse it to create a 1D test code
  - Reflects real applications
  - Small enough to fully dissect

$$\langle x \rangle = \frac{\int_0^\infty x\, e^{-x}\, dx}{\int_0^\infty e^{-x}\, dx} = 1$$

```
void kernel(FLOAT& x, FLOAT& p) {
   FLOAT xnew = drand()*FLOAT(23.0);
   FLOAT pnew = exp(-xnew);
   if (pnew > drand()*p) {
       x = xnew;
       p = pnew;
   }
}

p(x)=exp(-x)  --> probability
drand(x)      --> uniform ran# [0,1)
```

Sequential algorithm

- **N=240*256 independent samples**

```
FLOAT x[N], p[N], sum=0.0;                          initialize
for (int i=0; i<N; i++) {
      x[i] = urand()*FLOAT(23.0);
      p[i] = exp(-x[i]);
}
for (int iter=0; iter<NEQ; iter++) {
      for (int i=0; i<N; i++) {
            kernel(x[i], p[i]);                     equilibrate
      }
}
for (int iter=0; iter<NCOMPUTE; iter++) {
      for (int i=0; i<N; i++) {
            kernel(x[i], p[i]);
            sum += x[i];                            sample
      }
}
```


on x86 drand calls “standard” random() function

CUDA kernel

- **Parallelize over independent samples**

```
#define NBLOCK 240
#define NTHREAD 256
#define N NBLOCK*NTHREAD
const FLOAT fac = 1.0/(1ul<<32);
#define srand() unsigned int state = (blockIdx*NTHREAD + threadIdx.x)*3 + 1; \
                  for (int i=0; i<100; i++) drand();
#define drand() (state = 1103515245U*state + 12345U)*fac
__global__ void kernel(SUMTYPE* psum) {
    srand();
    FLOAT x = drand()*FLOAT(23.0), p = expf(-x);
    for (int iter=0; iter<NEQU; iter++) {
        FLOAT xnew = drand()*FLOAT(23.0), pnew = exp(-xnew);
        if (pnew > drand()*p) { x = xnew; p = pnew; }
    }
    SUMTYPE sum = 0.0;
    for (int iter=0; iter<NCOMPUTE; iter++) {
        FLOAT xnew = drand()*FLOAT(23.0), pnew = exp(-xnew);
        if (pnew > drand()*p) { x = xnew; p = pnew; }
        sum += x;
    }
    psum[blockIdx.x*NTHREAD + threadIdx.x] = sum;
}
```

call fast version as appropriate

drand calls crude LCG

optional different precision for accumulator

CUDA v.s. x86 sequential performance

Platform	Time/s	#cycles/core/iteration
Tesla C1060 + CUDA single precision	4.0	20.3
Intel i7 single core + GCC double precision	3410	145
Intel i7 single core + ICC double precision	2070	88

- **Tesla is running**
  - **850 x GCC and 520 x ICC**
- **This is a realistic and fair comparison of what is possible with minimal effort on both platforms**
- **But it tells us nothing about what is possible if we try hard**

gcc 4.4.0 with -O3 option icc 11 with -fast option

What's wrong on the Intel CPU?

- Is this a problem with the
  - compilers,
  - libraries,
  - algorithm design, or
  - an intrinsic failure of the Intel core?

  - Looking ahead, we will conclude that the failure is a team effort.
- We will see it is not due to
  - Hardware support for special functions on the GPU
  - Synergy between different GPU features, etc.

The poor, misunderstood x86 CPU

- It is not a sequential processor
  - It is a multi-issue, out-of-order, heavily pipe-lined device with SIMD acceleration & 4 cores
- Serial FP code will be too slow by up to ...
  - 4x pipeline latency (5x for i7 multiply?)
  - 2x SIMD register width (4x in single)
  - 2x simultaneous + and * issue
  - 4x for using only 1 out of 4 cores
  - Total is 4*2*2*4 = 64 (128 in single precision)
  - Need VL=80 in double precision (1 core)
    - "Fat" loops need shorter VL

Vectorizable?

```
void kernel(FLOAT& x, FLOAT& p) {
   FLOAT xnew = drand()*FLOAT(23.0);
   FLOAT pnew = exp(-xnew);
   if (pnew > drand()*p) {
      x = xnew;
      p = pnew;
   }
}
```

- This seems to epitomize sequential code
  - 2 rand#, 1 exp(), and 1 if-test
- But a 30-year old Cray compiler would have automatically vectorized this loop
  - manual inline and using ranf()

```
for (int i=0; i<N; i++)
   kernel(x[i], p[i]);
```

- So did the CUDA compiler NVCC
- But we're not aware of an x86 compiler that will oblige
  - Do it by hand

Hand vectorized kernel

- **Mix of VML and hand-coded SSE assembly**

```
double vkernel(int n, double* x, double* p) {
    static double xnew[VL] ALIGNED;
    static double pnew[VL] ALIGNED;
    static double test[VL] ALIGNED;
    vrand(n, test);
    vrand(n, xnew);
    vscale(n, -17.0, xnew, pnew);
    vexp(n, pnew, pnew);
    vmul(n, test, p, test);
    vDmask_lt(n, pnew, test, test);
    vDmerge(n, test, p, pnew, p);
    vDmerge(n, test, x, xnew, x);
    return vsum(n, x);
}
```

vectorized version of Brent's 48-bit LFG,

Uses CMPDD to form mask

Uses ANDPD, ANDNPD, POR to merge vectors under mask

VML is missing vectorized sum, mask, merge, and fast ran# generator
VML exp is both fast and accurate in LA mode ... EP is unreliable

Performance prediction and measurement

Operation	Cycles
2 random values	5
1 scale	0.5
1 exp	11
1 mul	0.5
1 compare	0.5
2 merge	4
1 sum	0.5
Total	21

- **Measured is 23.5 cycles/iteration/core**
- **Single precision will be exactly 2x as fast since it just involves 2x wider registers**
  - **Did not measure since am totally fed up of writing asm**

Bottom line

- **Head-to-head comparison of vectorized single precision kernels – Tesla:Intel quad core 17.6 : 1**
  - **~5x speed up in special function evaluation**
    - **Intel VML takes 11 cycles/exp in dp – GCC takes 56 in dp**
  - **2x speed up by comparing single to single**
  - **4x speed up by using all 4 cores**
- **The optimal x86 and CUDA kernels are "identical"**
  - **Loop over blocks to provide coarse parallelism and tile data into registers/cache/shared memory**
  - **Loop over threads/vector elements to provide SIMD parallelism and hide memory latency**
- **Any credible *architecture* benchmark must back port the CUDA kernel to the x86 & vectorize it**

Kudos to CUDA

- SIMT + rigid register and shmem constraints
  - Forces efficient program design for algorithms that fit
- NVIDIA GPGPU
  - Predicated ops + SFU ease code generation
- x86 compilers ignoring 25 years of knowledge?
  - 16-byte alignment problem (not in AMD SSE-128)
  - Cray is developing a vectorizing x86 compiler
- The CUDA code structure is optimal even for x86
- MCDUA (Stratton and Hwu)
  - Portable solution ... needs full vectorization not just serial loop plus intrinsics

MADNESS

Multiresolution
Adaptive
Numerical
Scientific
Simulation

Multiresolution Adaptive Numerical Scientific Simulation

Ariana Beste[1], George I. Fann[1], Robert J. Harrison[1,2], Rebecca Hartman-Baker[1], Jun Jia[1], Shinichiro Sugiki[1]
[1]Oak Ridge National Laboratory, [2]University of Tennessee, Knoxville

Gregory Beylkin[4], Fernando Perez[4], Lucas Monzon[4], Martin Mohlenkamp[5] and others
[4]University of Colorado, [5]Ohio University

Hideo Sekino[6] and Takeshi Yanai[7]
[6]Toyohashi University of Technology, [7]Institute for Molecular Science, Okazaki

harrisonrj@ornl.gov

SciDAC
Scientific Discovery through Advanced Computing

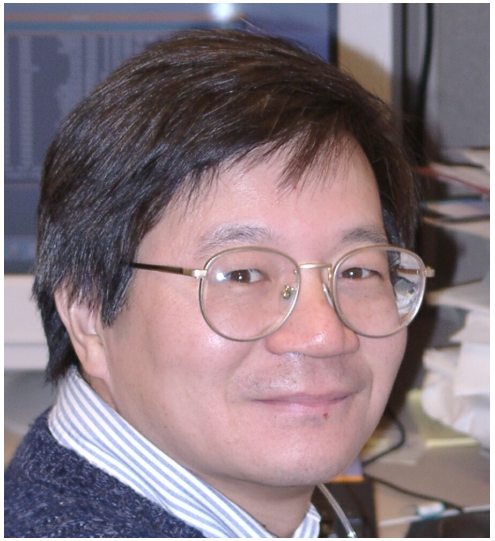
UT-BATTELLE
Oak Ridge National Laboratory

Multiresolution chemistry objectives

- Scaling to 1+M processors ASAP
- Complete elimination of the basis error
  - One-electron models (e.g., HF, DFT)
  - Pair models (e.g., MP2, CCSD, …)
- Correct scaling of cost with system size
- General approach
  - Readily accessible by students and researchers
  - Higher level of composition
  - Direct computation of chemical energy differences
- New computational approaches
  - *Fast algorithms with guaranteed precision*

The mathematicians …

Gregory Beylkin
http://amath.colorado.edu/faculty/beylkin/



George I. Fann
fanngi@ornl.gov

Essential techniques for fast computation

- Multiresolution

$$V_0 \subset V_1 \subset \cdots \subset V_n$$

$$V_n = V_0 + \left(V_1 - V_0\right) + \cdots + \left(V_n - V_{n-1}\right)$$

- Low-separation rank

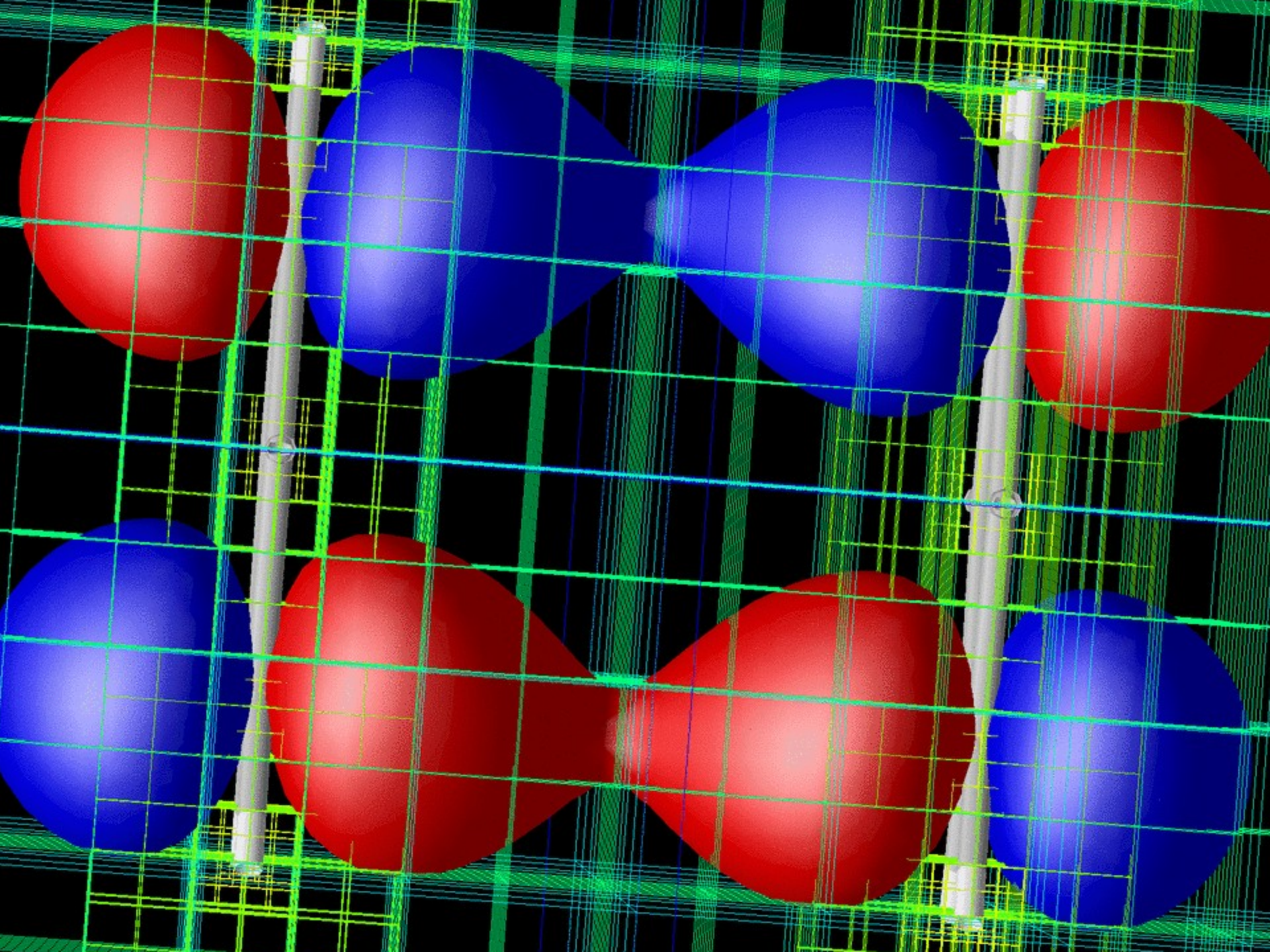
$$f(x_1, \ldots, x_n) = \sum_{l=1}^{M} \sigma_l \prod_{i=1}^{d} f_i^{(l)}(x_i) + O(\epsilon)$$

$$\left\| f_i^{(l)} \right\|_2 = 1 \qquad \sigma_l > \cdots$$

- Low-operator rank

$$A = \sum_{\mu=1}^{r} u_\mu \sigma_\mu v_\mu^T + O(\epsilon)$$

$$\sigma_\mu > 0 \qquad v_\mu^T v_\lambda = u_\mu^T u_\lambda = \delta_{\mu\nu}$$



Please forget about wavelets

- They are not central
- Wavelets are a convenient basis for spanning V_n-V_{n-1} and understanding its properties
- But you don't actually need to use them
  - MADNESS does still compute wavelet coefficients, but *Beylkin's new code does not*
- Please remember this …
  - Discontinuous spectral element with multi-resolution and separated representations for fast computation with guaranteed precision in many dimensions.

Computational kernels

- Discontinuous spectral element
  - In each “box” a tensor product of coefficients
  - Most operations are small matrix-multiplication

$$r_{i'j'k'} = \sum_{ijk} s_{ijk} c_{ii'} c_{jj'} c_{kk'} = \sum_{k} \left(\sum_{j} \left(\sum_{i} s_{ijk} c_{ii'} \right) c_{jj'} \right) c_{kk'}$$

$$\Rightarrow r = \left(\left(s^T c \right)^T c \right)^T c$$

  - Typical matrix dimensions are 2 to 30
  - E.g., $(20,400)^T$ * (20,20)

Ratio of Speeds of MKL, Goto, ATLAS to MTXMQ on Intel Xeon 5355 for $(20,400)^T*(20,n)$.

n	MKL	Goto	ATLAS		n	MKL	Goto	ATLAS
2	6.25	4.1667	5		16	0.8966	1.2581	2.0708
4	3.1042	3.6341	4.6563		18	1.7763	1.3636	2.4545
6	4.375	2.625	5.122		20	0.9556	1.2727	2.6168
8	1.3132	2.0427	5.1957		22	1.6416	1.2968	2.7308
10	2.7368	1.9549	5.3061		24	0.9638	1.2208	1.9664
12	1.0605	1.5843	2.4352		26	1.5337	1.2814	2.1295
14	2.0323	1.4737	2.1356		28	0.8411	1.0588	2.0301

XT5 single core FLOPs/cycle

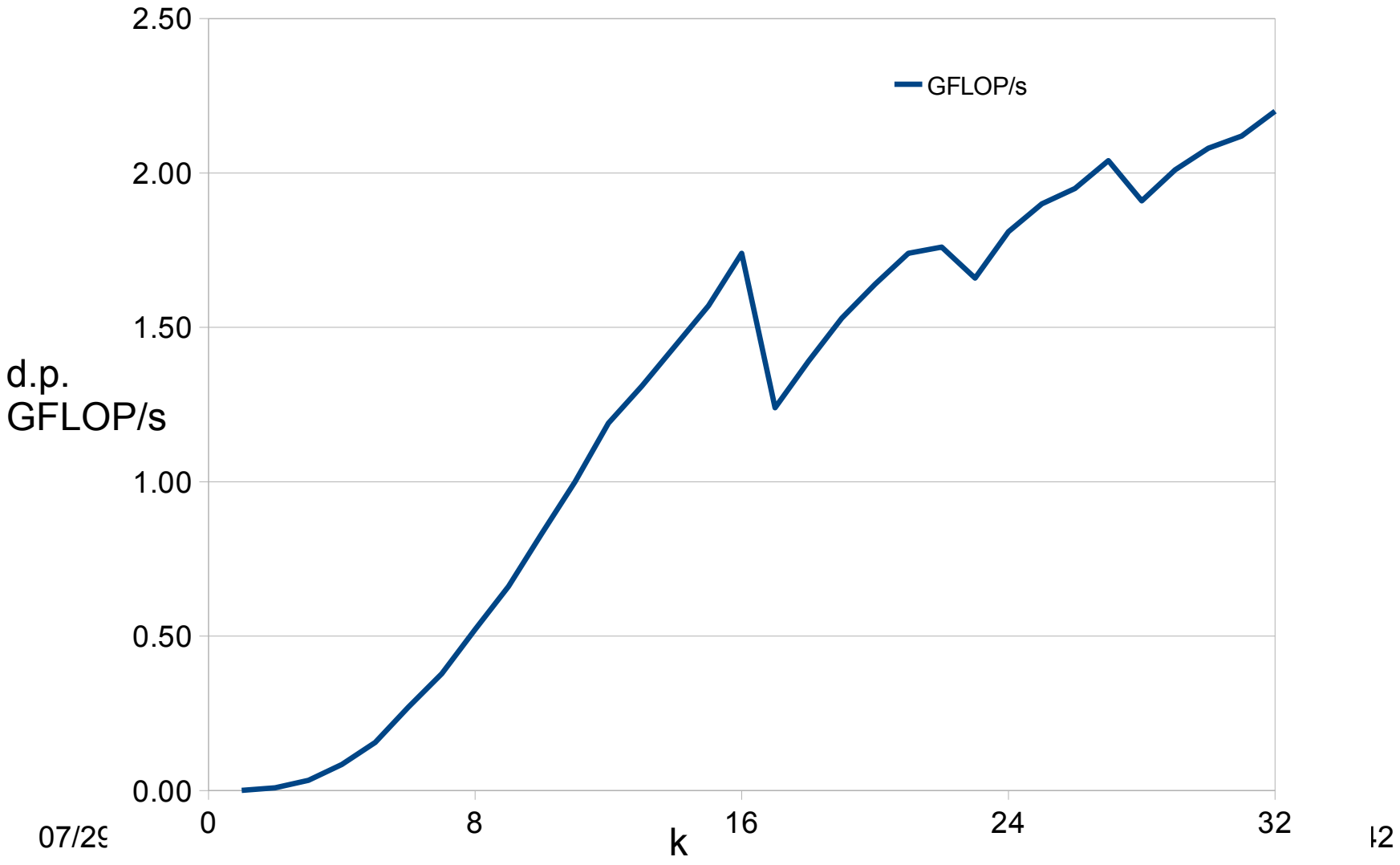
(nj, ni)T*(nj,nk)				
ni	nj	nk	MTXMQ	ACML
400	2	20	2.55	0.95
400	4	20	2.62	1.50
400	6	20	2.60	1.79
400	8	20	2.56	2.02
400	10	20	2.58	2.12
400	12	20	2.64	2.27
400	14	20	2.90	2.35
400	16	20	2.80	2.46
400	18	20	2.74	2.49
400	20	20	2.89	2.58

nested transform (nj, ni)T*(nj,nk)				
ni	nj	nk	MTXMQ	ACML
4	2	2	0.10	0.07
16	4	4	1.04	0.51
36	6	6	1.74	0.99
64	8	8	2.33	1.56
100	10	10	2.61	1.80
144	12	12	2.69	2.12
196	14	14	2.94	2.17
256	16	16	2.97	2.41
324	18	18	2.93	2.38
400	20	20	3.03	2.49
484	22	22	3.01	2.52
576	24	24	3.09	2.73
676	26	26	3.02	2.73
784	28	28	2.87	2.87
900	30	30	2.88	2.81

L2 cache is 512Kb = 2*32^3 doubles
- hence expect good multi-core scaling
- measured linear speed up all 8 cores

Initial results C1060 Single MP

$M^T x M \quad [k,k^2]^T x [k,k]$



Combine multiple small kernels

- Task queue implemented on Tesla
  - Each task targets 1 MP (1 multi-threaded block)
  - In principle C++ templates should work, but ??
  - Overlap data transfer with compute
  - Looking forward to next gen. card rumored to have more MIMD capabilities
- E.g., model kernel of convolution operator

$$r = \sum_{\mu} \left(\left(s^T X^{(\mu)} \right)^T Y^{(\mu)} \right)^T Z^{(\mu)}$$

  - 60 GF/s double precision ([32,1024]*[32,32]))
  - But this is neglecting many optimizations ... currently projecting perhaps 40 GF/s

Applications under development

- DFT & HF for electrons
  - Energies, gradients, spectra, non-linear optical properties, Raman intensities (Harrison, Sekino, Yanai Vasquez)
  - Molecules & periodic systems (Eguilez and Thornton)
- Atomic and molecular physics
  - Exact dynamics of few electron systems in strong fields (Krstic and Vence), MCSCF for larger systems
- Nuclear structure
  - G. Fann, et al.
- Preliminary studies in fusion and climate

Dynamics of H_2^+ in laser

- 3D
- fixed nuclei

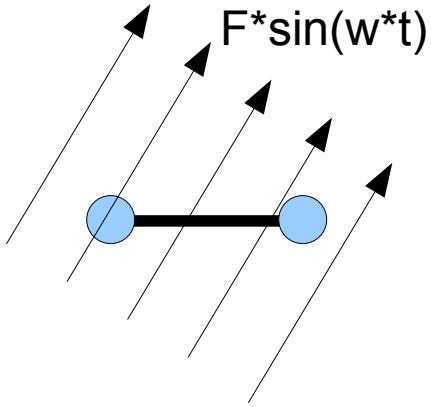
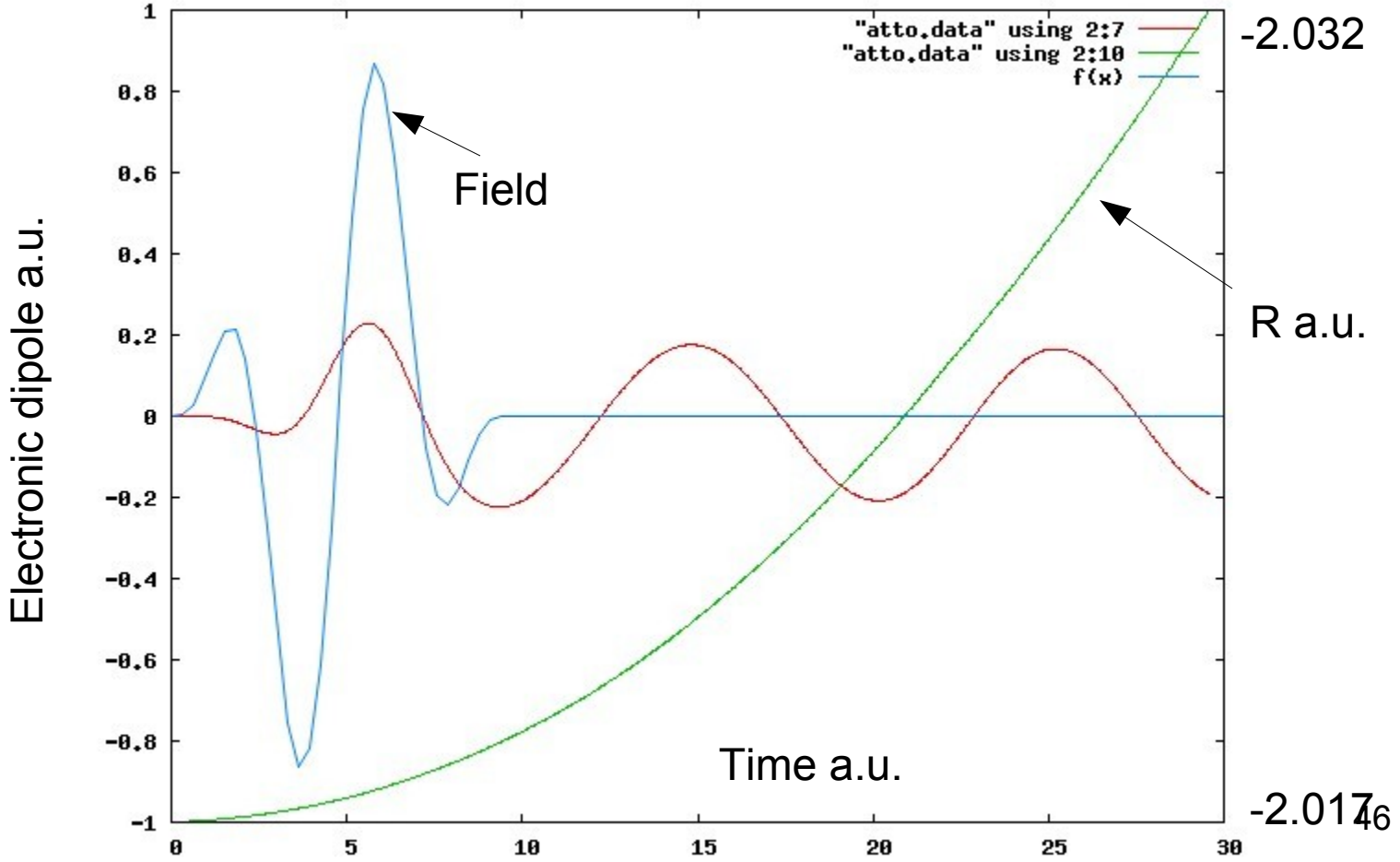


Image shows electron density evolving in collinear field but multiple angles have been simulated

log10 scale, [-30,30]^3

Dynamics of H_2^+ in laser

- 4D – 3 electronic + internuclear coordinate
  - First simulation with quantum nuclei and non-collinear field (field below is transverse)



High-level composition

- Close to the physics

$$E = \langle \psi | -\frac{1}{2}\nabla^2 + V | \psi \rangle + \int \psi^2(x) \frac{1}{|x-y|} \psi^2(y)\, dx\, dy$$

```
operatorT op = CoulombOperator(k, rlo, thresh);
functionT rho = psi*psi;
double twoe = inner(apply(op,rho),rho);
double pe = 2.0*inner(Vnuc*psi,psi);
double ke = 0.0;
for (int axis=0; axis<3; axis++) {
    functionT dpsi = diff(psi,axis);
    ke += inner(dpsi,dpsi);
}
double energy = ke + pe + twoe;
```

High-level composition

- Express ***ALL*** available parallelism without burdening programmer
  - Internally, MADNESS is looking after data and placement and scheduling of operations on individual functions
  - Programmer must express parallelism over multiple functions and operators
    - But is *not* responsible for scheduling or placement

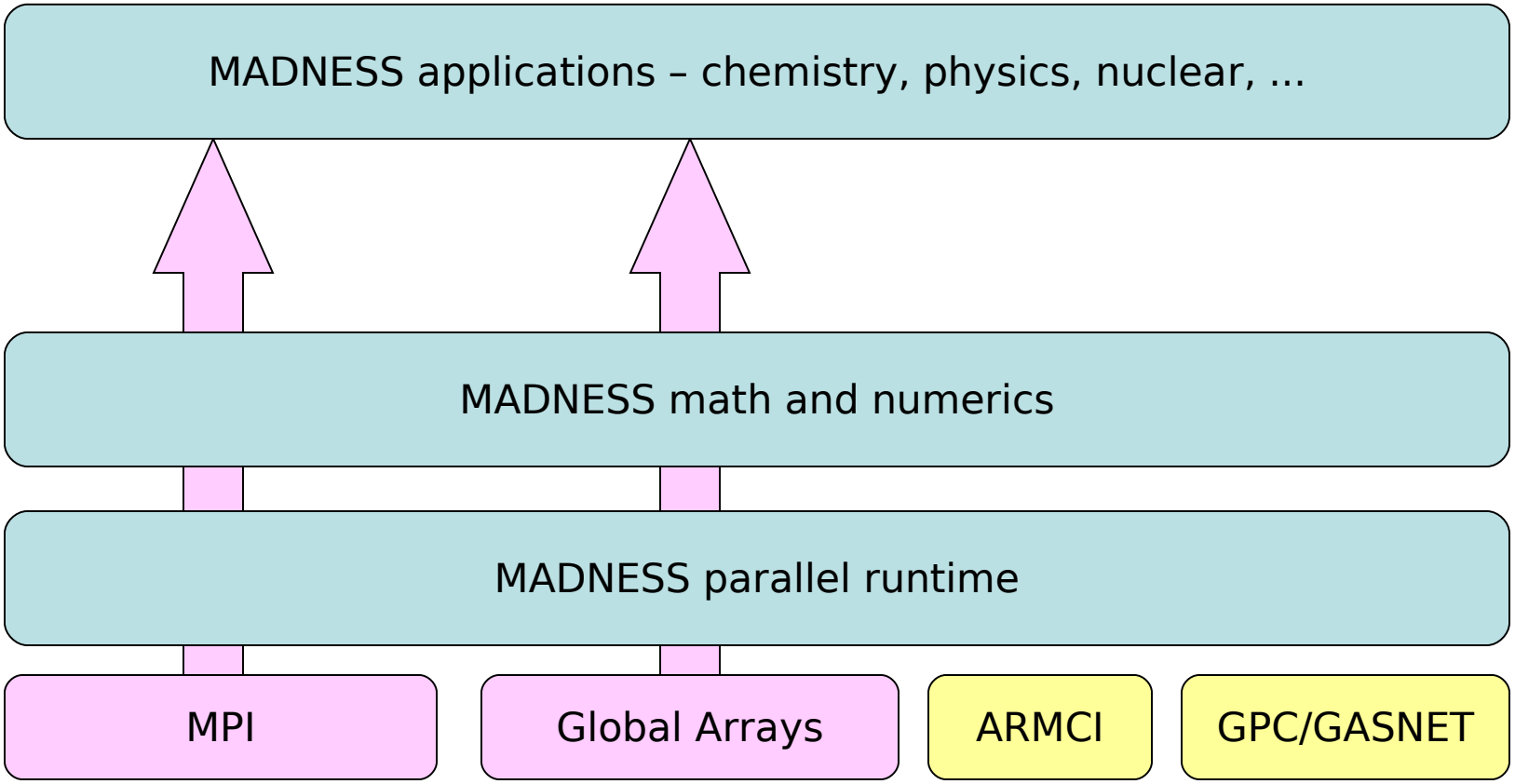
High-level composition

- E.g., make the matrix of KE operator
  - All scalar operations include optional fence
    - E.g., functionT scale(const functionT& f, T scale, bool fence=true)
  - Internally, operations on vectors schedule all tasks with only one fence

$$\langle \phi_i | -\frac{1}{2}\nabla^2 | \phi_j \rangle = \frac{1}{2}\langle \nabla^T \phi_i \nabla \phi_j \rangle$$

```
Tensor<double>
kinetic_energy_matrix(World& world,
            const vector<functionT>& v) {
  int n = v.size();
  Tensor<double> r(n,n);
  for (int axis=0; axis<3; axis++) {
     vector<functionT> dv = diff(world,v,axis);
     r += inner(world, dv, dv);
  }
  return r.scale(0.5);
}
```

MADNESS architecture

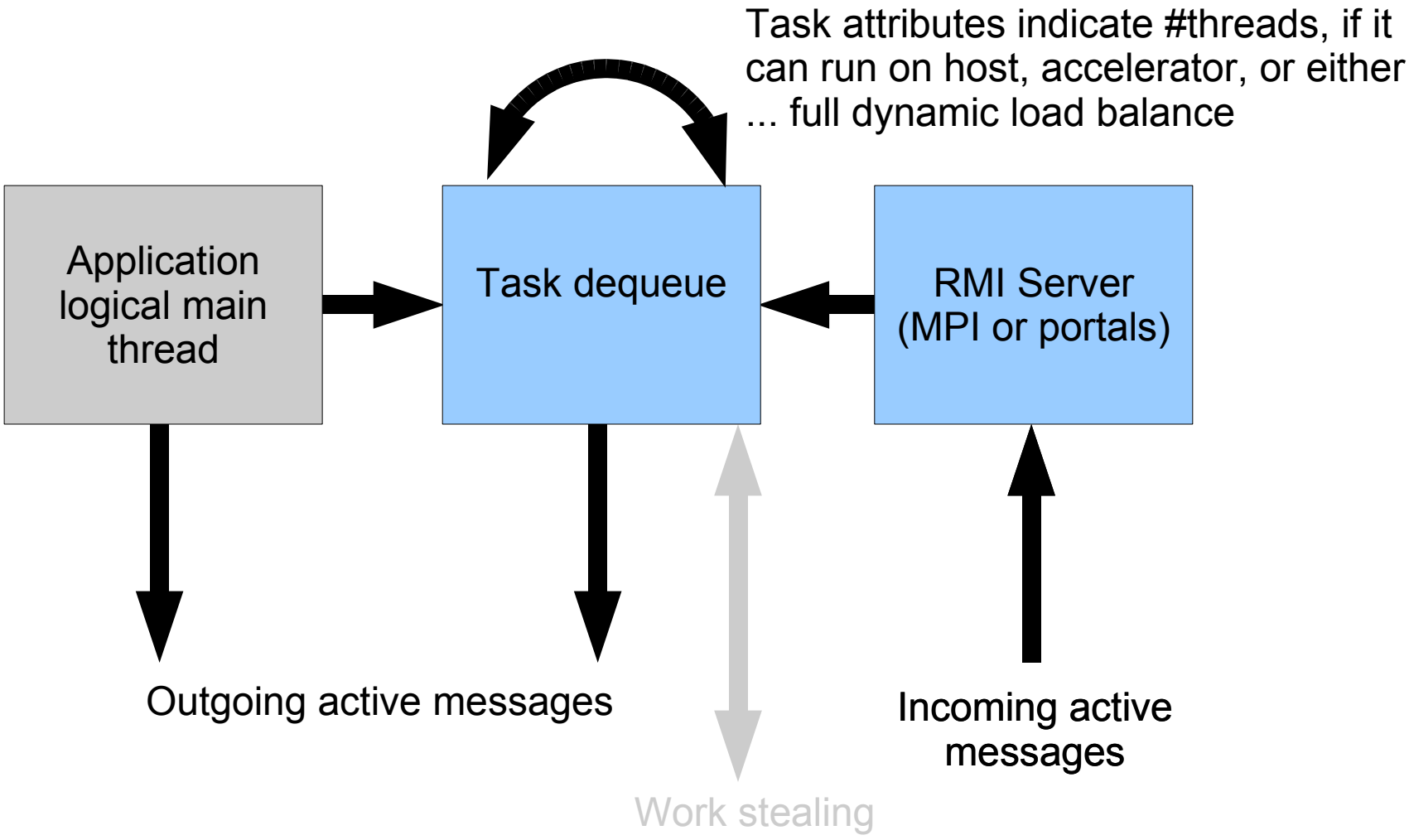


Intel Thread Building Blocks being considered for multicore

Runtime Objectives

- Scalability to 1+M processors ASAP
- Runtime responsible for
  - scheduling and placement,
  - managing data dependencies,
  - hiding latency, and
  - Medium to coarse grain concurrency
- Compatible with existing models
  - MPI, Global Arrays
- Borrow successful concepts from Cilk, Charm++, Python
- Anticipating next gen. languages

Multi-threaded architecture



Key elements

- Futures for hiding latency and automating dependency management
- Global names and name spaces
- Non-process centric computing
  - One-sided messaging between objects
  - Retain place=process for MPI/GA legacy
- Dynamic load balancing
  - Data redistribution, work stealing, randomization

Futures

- Result of an asynchronous computation
  - Cilk, Java, HPCLs
- Hide latency due to communication or computation
- Management of dependencies
  - Via callbacks

```
int f(int arg);
ProcessId me, p;

Future<int> r0=task(p, f, 0);
Future<int> r1=task(me, f, r0);

// Work until need result

cout << r0 << r1 << endl;
```

Process “me” spawns a new task in process “p” to execute `f(0)` with the result eventually returned as the value of future `r0`. This is used as the argument of a second task whose execution is deferred until its argument is assigned. Tasks and futures can register multiple local or remote callbacks to express complex and dynamic dependencies.

Global Name Spaces

- Specialize global names to containers
  - Hash table done
  - Arrays, etc., planned
- Replace global pointer (process+local pointer) with more powerful concept
-
- User definable map from keys to “owner” process

```
class Index;  // Hashable
class Value {
   double f(int);
};

WorldContainer<Index,Value> c;
Index i,j;  Value v;
c.insert(i,v);
Future<double> r =
   c.task(j,&Value::f,666);
```

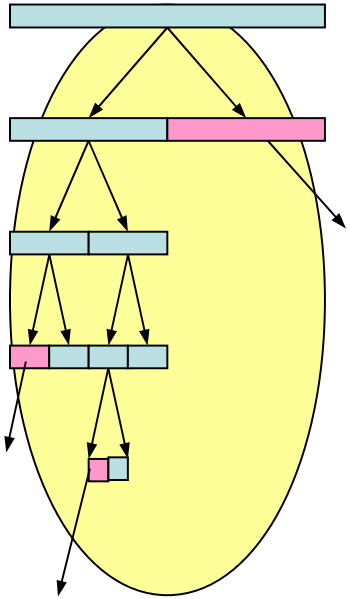
A container is created mapping indices to values.

A value is inserted into the container.

A task is spawned in the process owning key `j` to invoke `c[j].f(666)`.

Name spaces are a large part of the elegance of Python and success of Charm++ (chares+arrays).

Virtualization of data and tasks



Parameter:
- MPI rank
- probe()
- set()
- get()

Task:
- Input parameters
- Output parameters
- probe()
- run()

```
Future Compress(tree):
      Future left = Compress(tree.left)
      Future right = Compress(tree.right)
      return Task(Op, left, right)

Compress(tree)
Wait for all tasks to complete
```

Benefits: Communication latency & transfer time largely hidden
Much simpler composition than explicit message passing
Positions code to use "intelligent" runtimes with work stealing
Positions code for efficient use of multi-core chips
Positions code for efficient use of hybrid architectures

Summary

- Huge computational resources are rushing towards us
  - Tremendous scientific potential
  - Tremendous challenges in
    - Research,
    - Education, and
    - Community
- UT and ORNL
  - Think of us when you have/want something fun and challenging to do – or if you have some good students looking for challenging graduate or postdoctoral study

Acknowledgements

- Our work on accelerators is supported by the National Science Foundation, grant CHE 0625598
  - Cyber-infrastructure and Research Facilities: Chemical Computations on Future High-end Computers
- Work on MADNESS and NWChem is supported by the U.S. Department of Energy, the divisions of Advanced Scientific Computing Research and Basic Energy Science, Office of Science, under contract DE-AC05-00OR22725 with Oak Ridge National Laboratory
- This research was performed in part using resources of the National Center for Computational Sciences at Oak Ridge National Laboratory under contract DE-AC05-00OR22725
- Greg Peterson @ UT EECS and his talented students Akila Gothandaraman and Rick Weber