Parallel Quantum Molecular Dynamics

Aiichiro Nakano

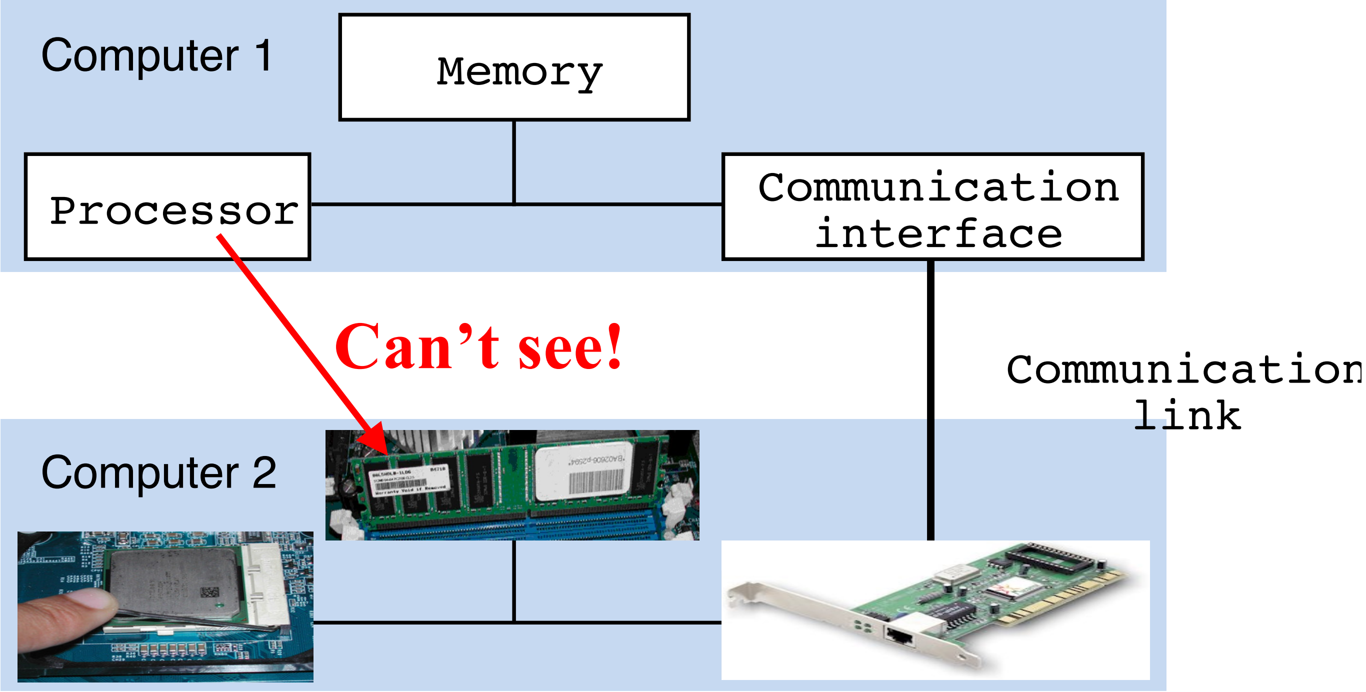
Collaboratory for Advanced Computing & Simulations
Department of Computer Science
Department of Physics & Astronomy
Department of Quantitative & Computational Biology
University of Southern California

Email: anakano@usc.edu

For basic & advanced parallel computing, see:

A. Grama *et al*., Introduction to Parallel Computing (Addison-Wesley, '03)
Extreme-scale-computing bootcamp at Argonne National Laboratory
https://aiichironakano.github.io/cs596.html

Parallel Computing Hardware

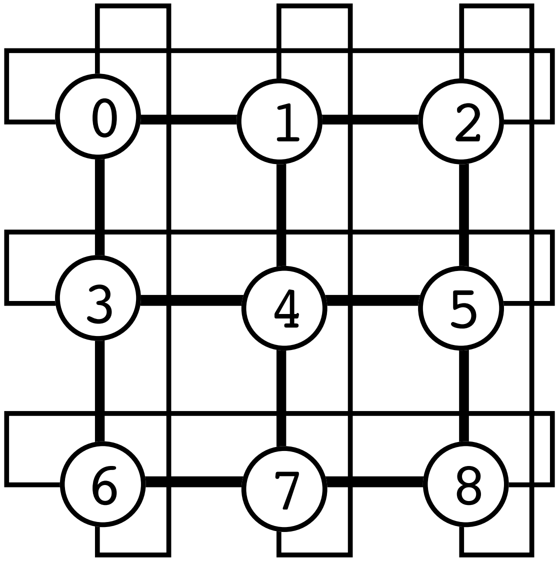


- **Processor: Executes arithmetic & logic operations**
- **Memory: Stores program & data (stored program computer)**
- **Communication interface: Performs signal conversion & synchronization between communication link & a computer**
- **Communication link: A wire capable of carrying a sequence of bits as electrical (or optical) signals**

See https://aiichironakano.github.io/cs596.html

Communication Network

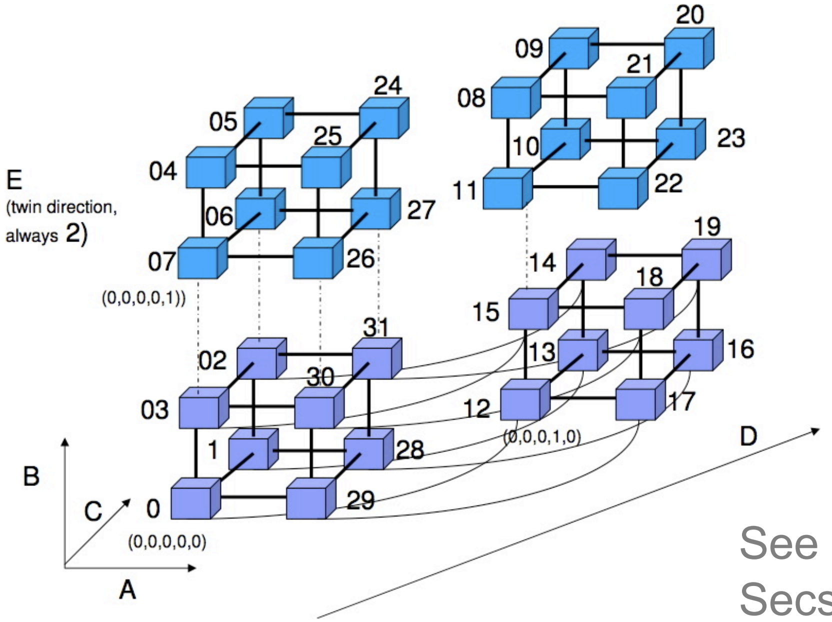
Mesh (torus)



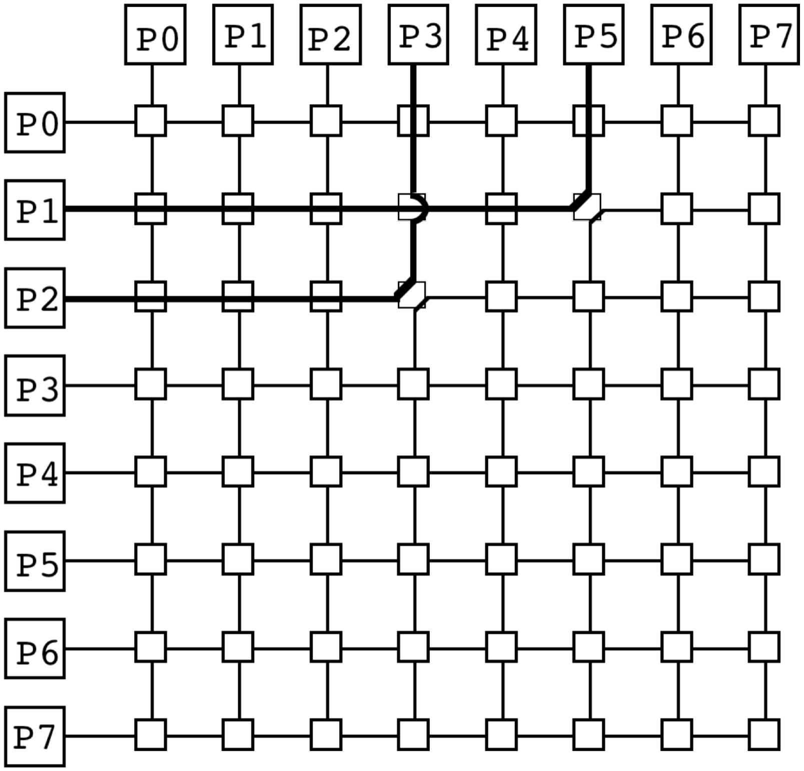
Wiring of interconnection network cables
Earth Simulator Research and Development Center

NEC Earth Simulator (640x640 crossbar)

IBM Blue Gene/Q (5D torus)



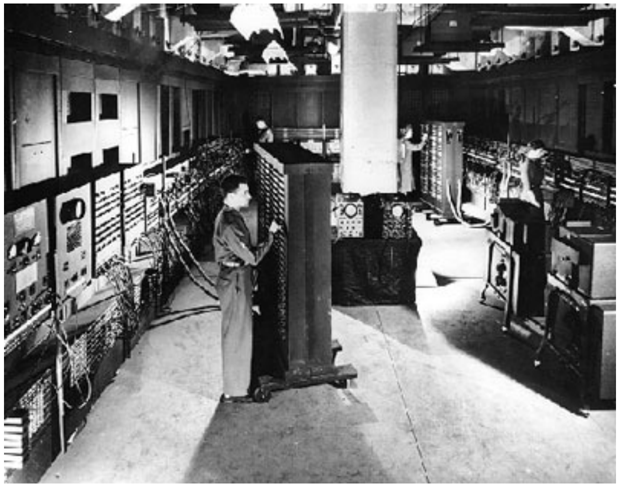
Crossbar switch



See Grama,
Secs. 2.4.2-2.4.4

History of Supercomputers

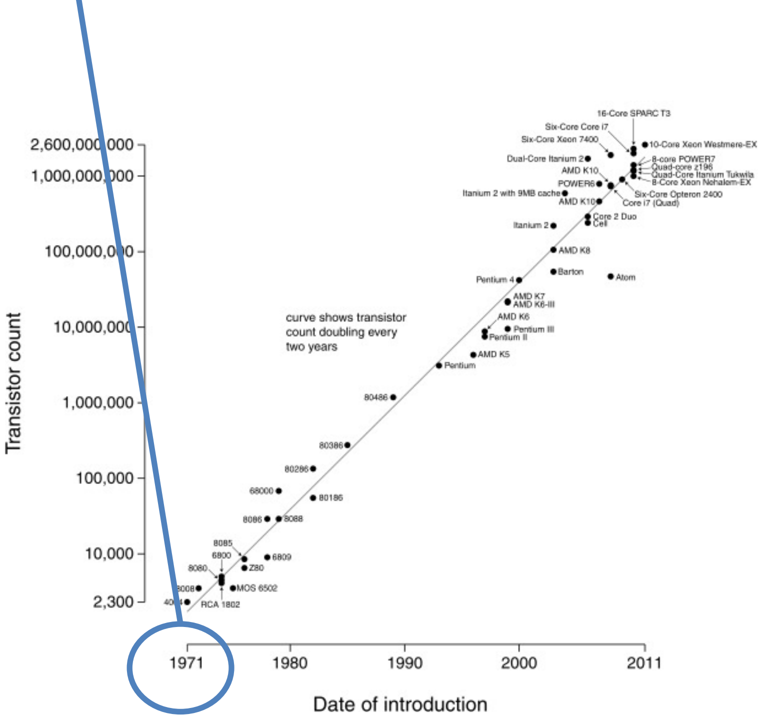
Early ’40s: ENIAC by Presper Eckert & John Mauchly at Univ. of Pennsylvania—First general-purpose electronic computer



’76: Cray 1 by Seymour Cray—beginning of vector supercomputer era

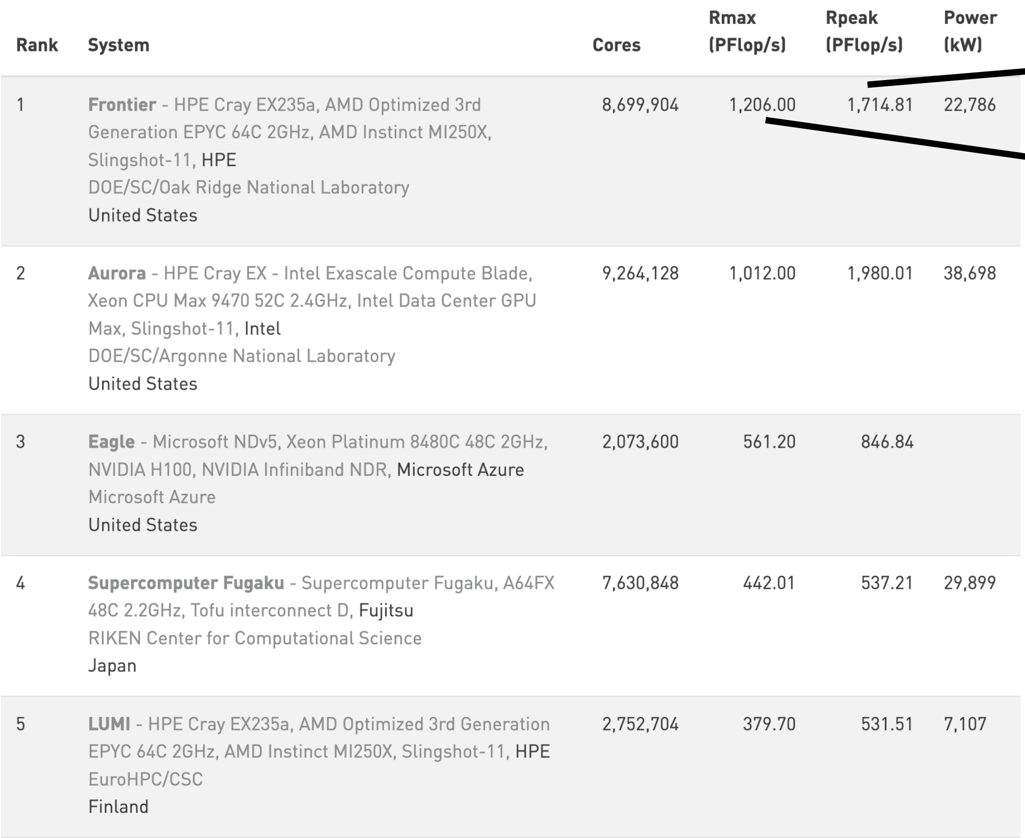
Late 80’s: massively parallel computers such as the Thinking Machines CM-2

(’71: Intel 4004—invention of microprocessor)



See lecture on MD machines

Merge of PC & Supercomputers



Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,206.00	1,714.81	22,786
2	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
3	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
4	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
5	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107

Theoretical performance

Measured performance (in Pflop/s)

Flop/s = floating-point operations/second

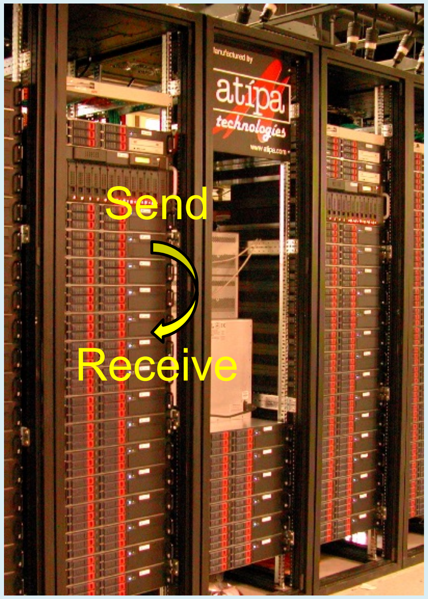
M (mega) = 10^6
G (giga) = 10^9
T (Tera) = 10^{12}
P (Peta) = 10^{15}
X (Exa) = 10^{18}

http://www.top500.org (June ’24)

- **USC-CARC: 30,000 cores**
- **CACS: 4,096 cores**
- **CACS-INCITE: 4M node-hours/year on exaflop/s Aurora at Argonne Nat’l Lab**

1.2 exaflop/s Frontier

MPI Programming

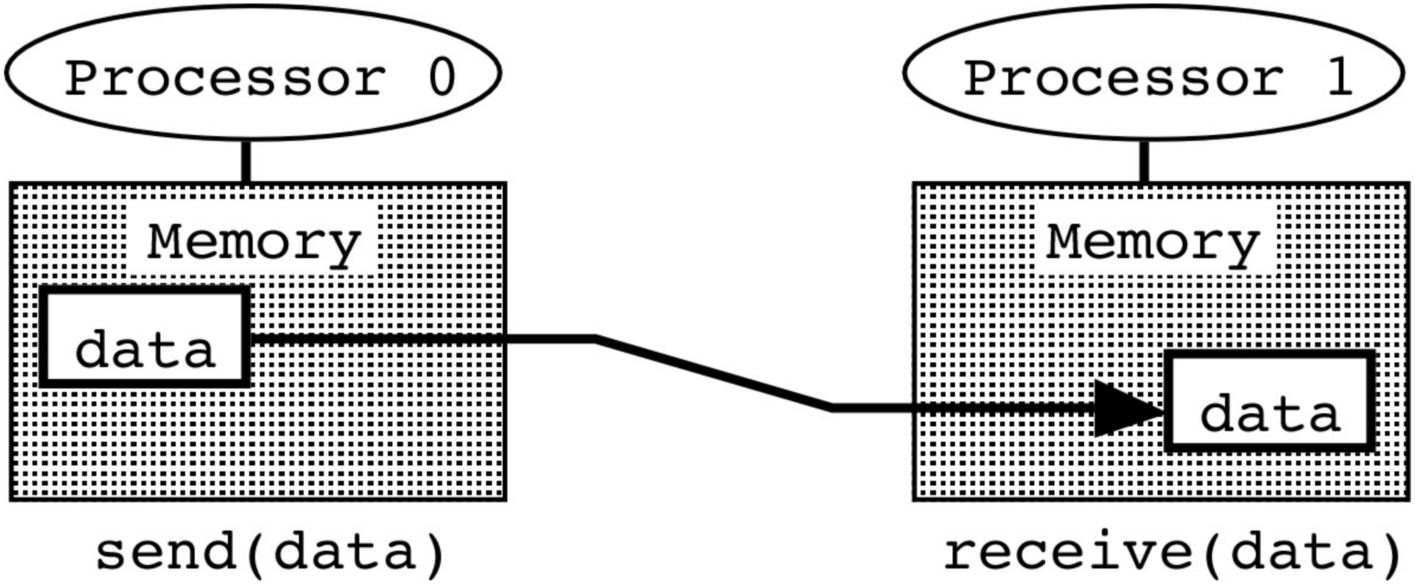


```
#include "mpi.h"
#include <stdio.h>
main(int argc, char *argv[]) {
  MPI_Status status;
  int myid;
  int n;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);
  if (myid == 0) {
    n = 777;
    MPI_Send(&n, 1, MPI_INT, 1, 10, MPI_COMM_WORLD);
  }
  else {
    MPI_Recv(&n, 1, MPI_INT, 0, 10, MPI_COMM_WORLD, &status);
    printf("n = %d\n", n);
  }
  MPI_Finalize();
}
```

Only need two (send & receive) functions!

https://www.mcs.anl.gov/research/projects/mpi

Single Program Multiple Data (SPMD)



Process 0

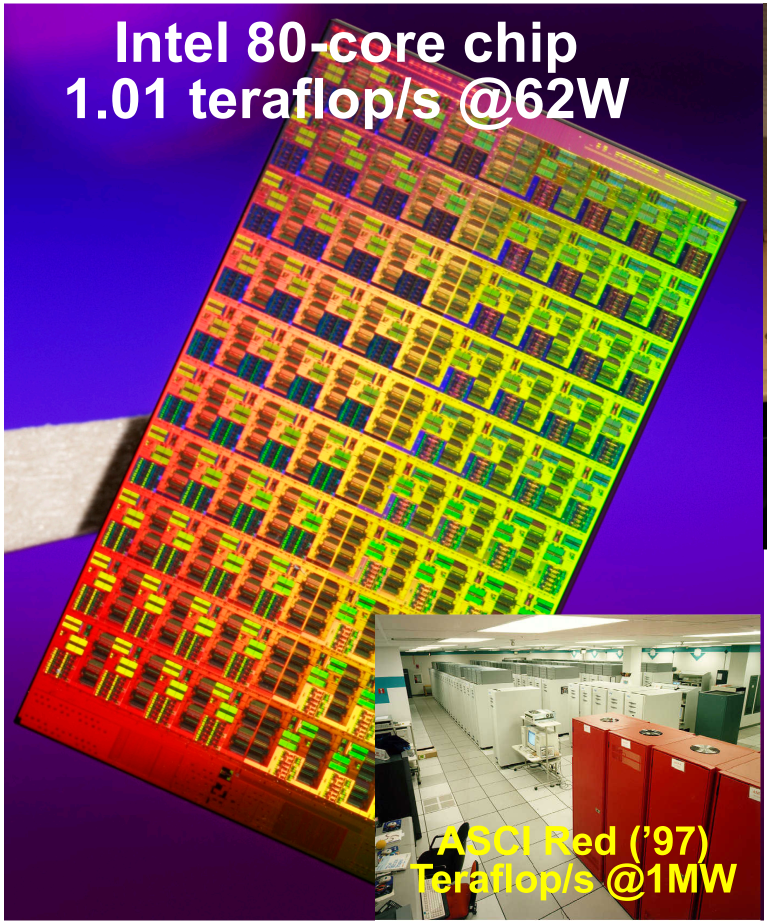
```
if (myid == 0) {
  n = 777;
  MPI_Send(&n,...);
}
else {
  MPI_Recv(&n,...);
  printf(...);
}
```

Process 1

```
if (myid == 0) {
  n = 777;
  MPI_Send(&n,...);
}
else {
  MPI_Recv(&n,...);
  printf(...);
}
```

Parallel computing: Specifies "Who does what"

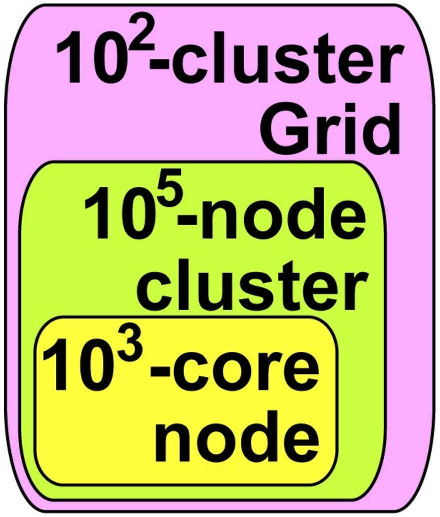
Many-core CPU/GPU Computing



Intel 80-core chip
1.01 teraflop/s @62W

ASCI Red ('97)
Teraflop/s @1MW

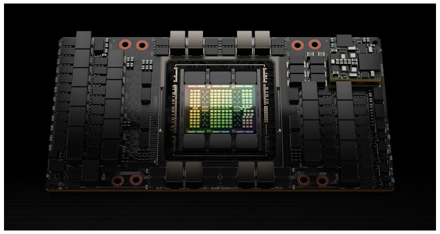
CACS PS3 cluster



10^2-cluster Grid

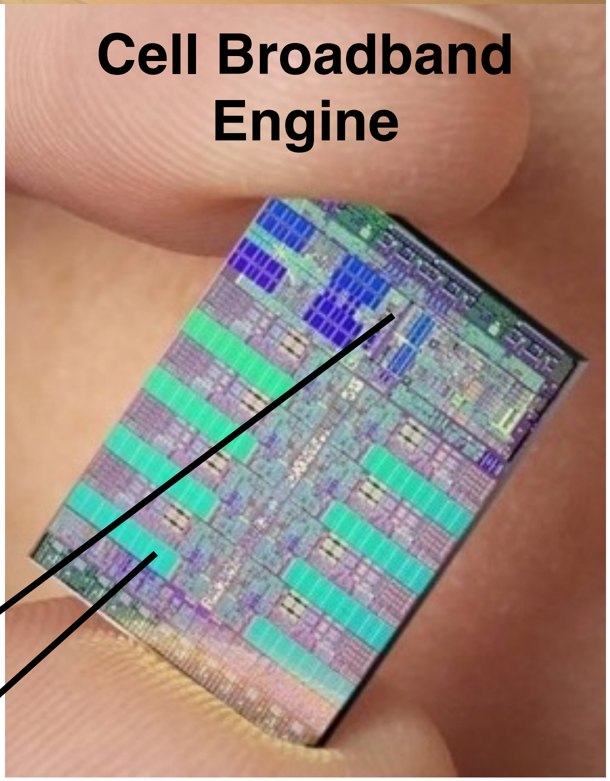
10^5-node cluster

10^3-core node



14,592 cores
67 teraflop/s

NVIDIA H100

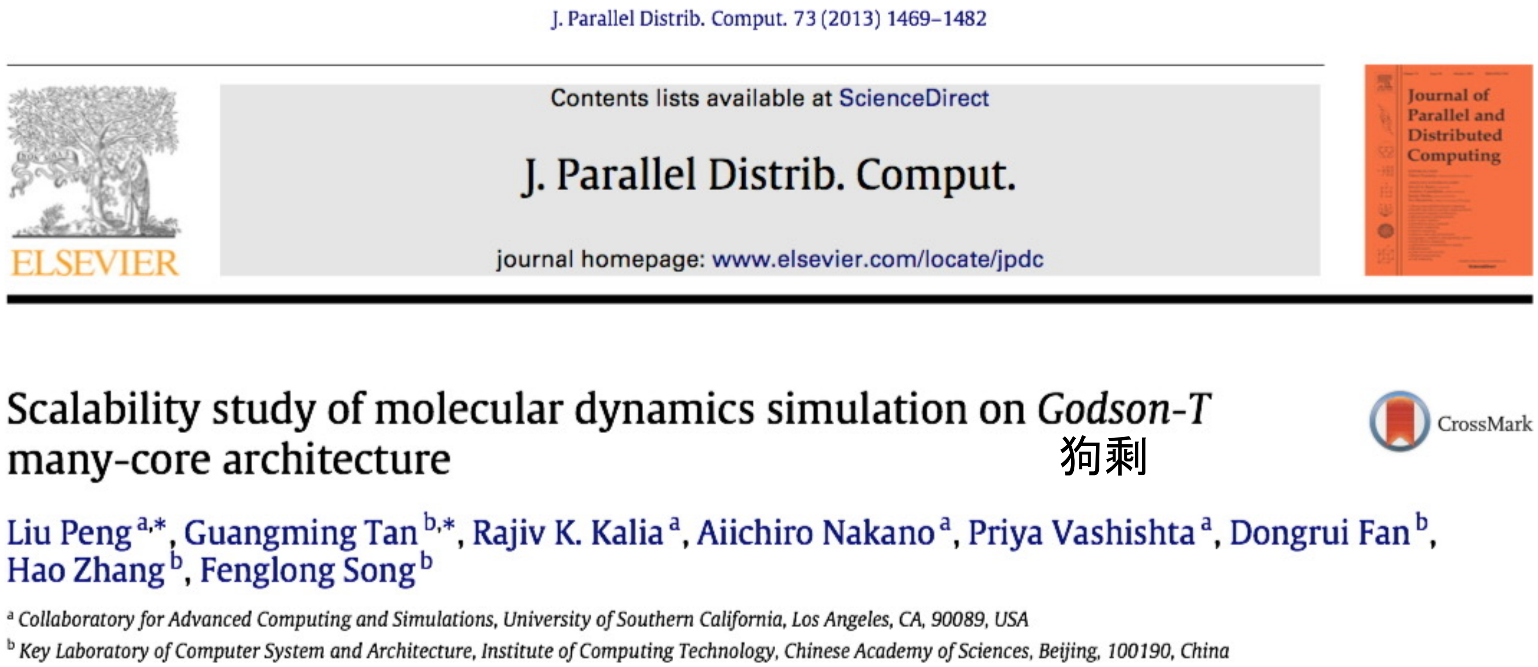


Cell Broadband Engine

64bit PowerPC

8 synergistic processing elements

Godson-T Many-core Architecture



J. Parallel Distrib. Comput. 73 (2013) 1469–1482

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

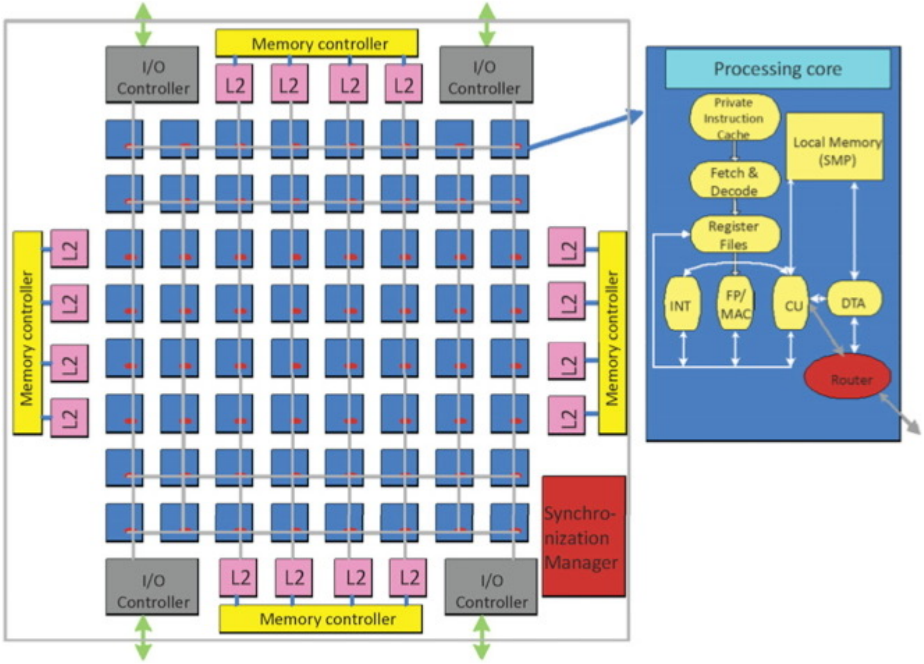
Scalability study of molecular dynamics simulation on *Godson-T* many-core architecture

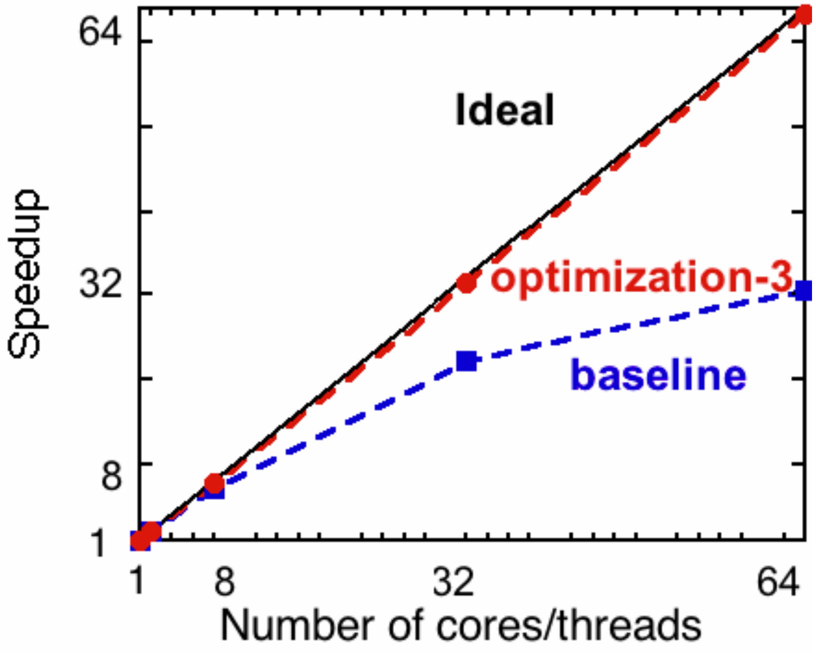
狗剩

Liu Peng[a,*], Guangming Tan[b,*], Rajiv K. Kalia[a], Aiichiro Nakano[a], Priya Vashishta[a], Dongrui Fan[b], Hao Zhang[b], Fenglong Song[b]

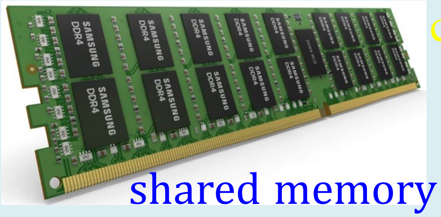
[a] Collaboratory for Advanced Computing and Simulations, University of Southern California, Los Angeles, CA, 90089, USA

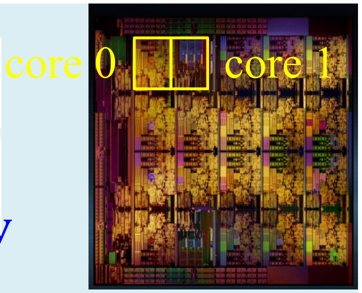
[b] Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, 100190, China





OpenMP Programming





```
#include <stdio.h>
#include <omp.h>
void main () {
  int nthreads,tid;
  nthreads = omp_get_num_threads();
  printf("Sequential section: # of threads = %d\n",nthreads);
  /* Fork multi-threads with own copies of variable */
  #pragma omp parallel private(tid)
  {
    /* Obtain & print thread id */
    tid = omp_get_thread_num();
    printf("Parallel section: Hello world from thread %d\n",tid);
    /* Only master thread does this */
    if (tid == 0) {
      nthreads = omp_get_num_threads();
      printf("Parallel section: # of threads = %d\n",nthreads);}
  } /* All created threads terminate */
}
```

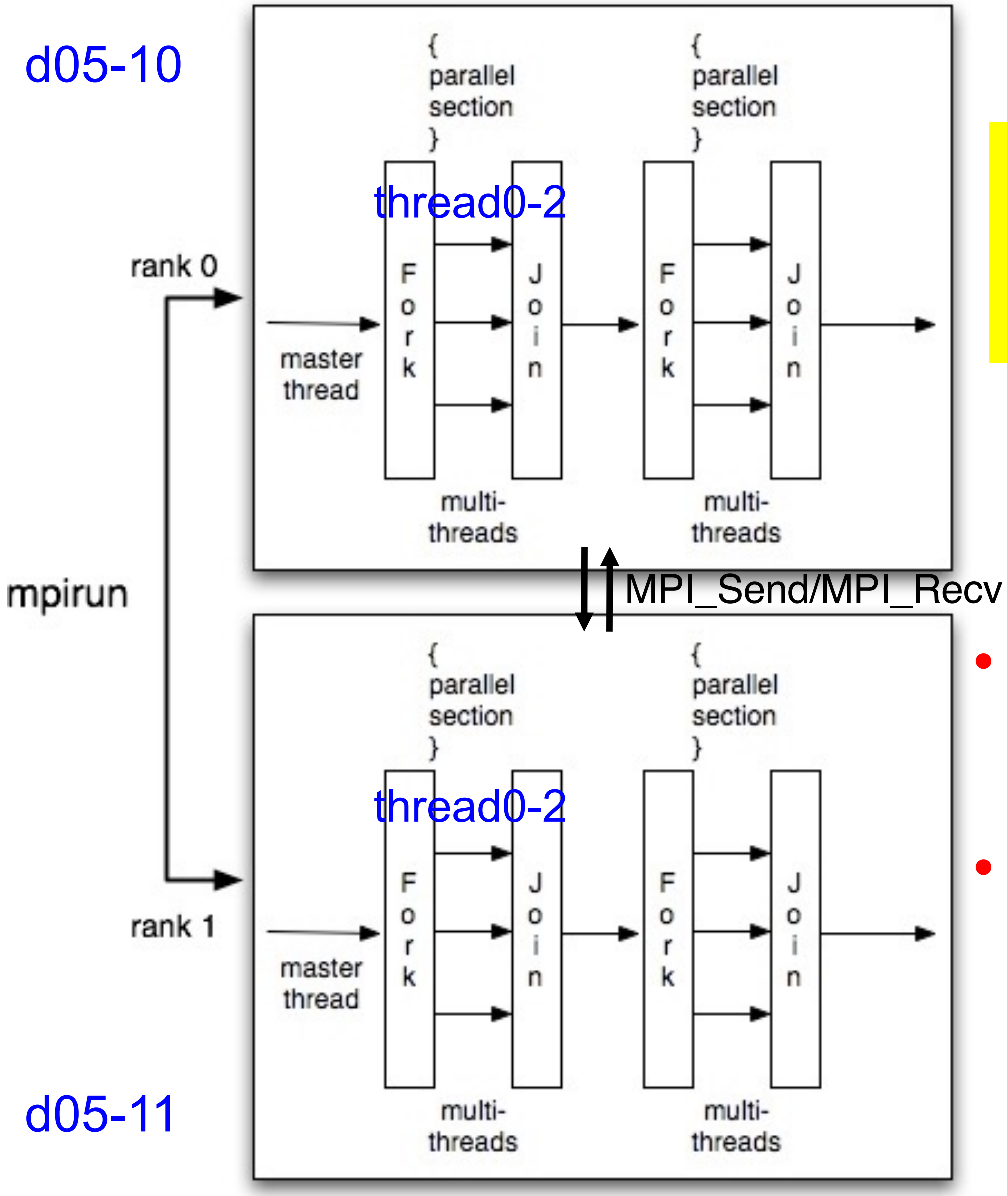
parallel section

- **Obtain the number of threads & my thread ID**
- **By default, all variables are shared unless selectively changing storage attributes using private clauses**

http://www.openmp.org

Hybrid MPI+OpenMP Programming

Each MPI process spawns multiple OpenMP threads



```
In a Slurm script:
  mpirun –n 2
In the code:
  omp_set_num_threads(3);
```

- **MPI processes communicate by sending/receiving messages**
- **OpenMP threads communicate by writing to/reading from shared variables**

SIMD Vectorization: MD

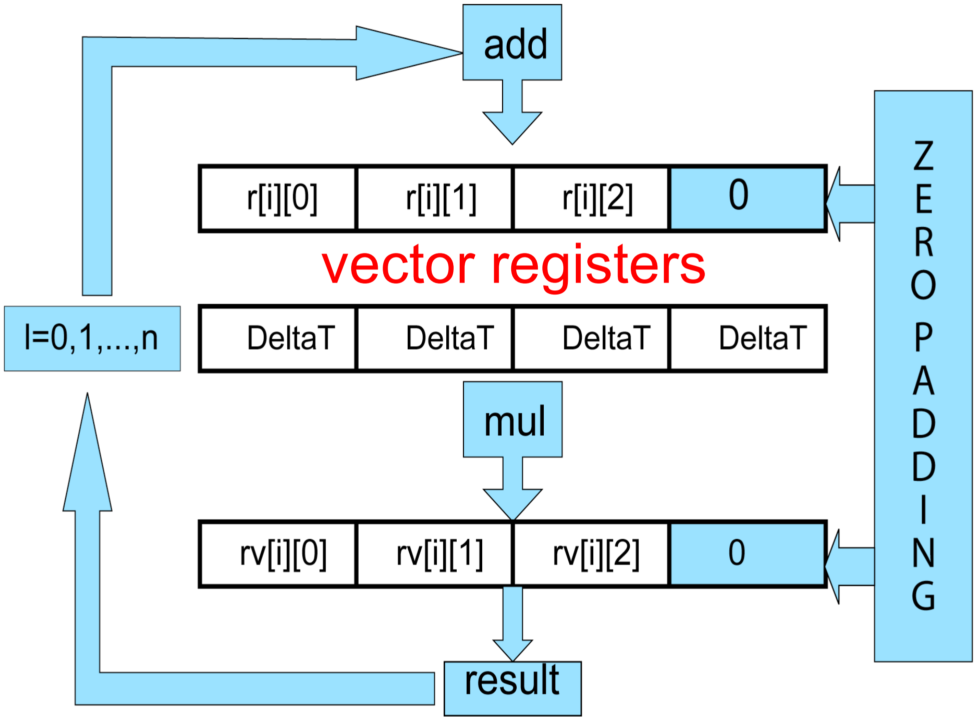
- **Single-instruction multiple-data (SIMD) parallelism using vector registers**

(Example) Zero padding to align complex data in molecular dynamics

Original solution

```
for (i=0; i<N; i++)
  for (a=0; a<3; a++)
    r[i][a] =
    r[i][a] +
    DeltaT*rv[i][a];
```

SIMD solution



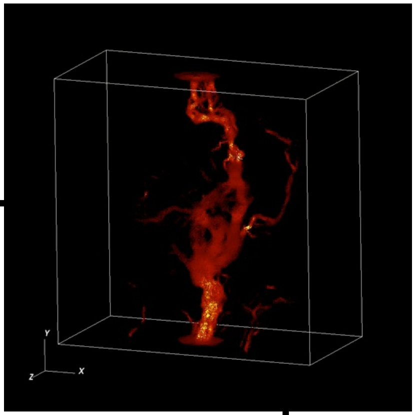
Peng *et al*., *PDPTA 2009*; *UCHPC 2010; J. Supercomputing* **57**, 20 ('11)

SIMD Vectorization: LBM

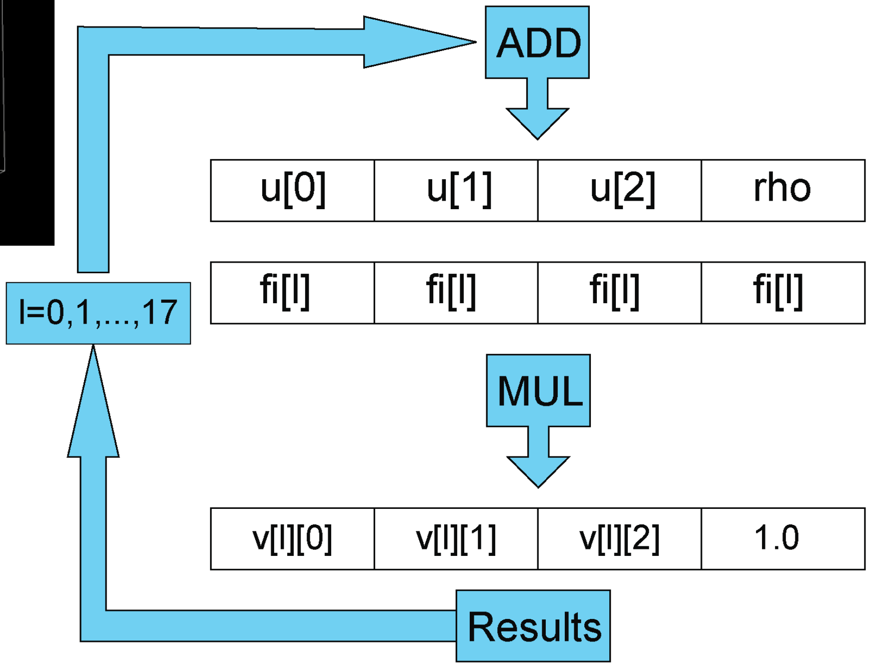
- **Translocated statement fusion in lattice-Boltzmann (LBM) flow simulation**

Original solution

```
for(i=0;i<3;i++){
  u[i]=0.0; rho=0.0;
  for(l=0;l<18;l++){
    fi[l] = f[18*cnz+l];
    u[i] += fi[l]*v[l][i];
    rho += fi[l];
  }
}
```



SIMD solution



3×18×5 = 270 computation

18×4 = 72 computation

Ideal speedup: 3.5

SIMDizable mathematical formulations:
Special relativity, quaternion, *etc.*

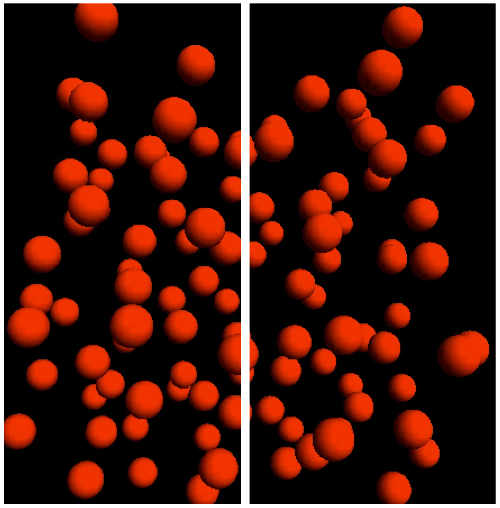
$$J^\alpha = (c\rho, j^1, j^2, j^3)$$

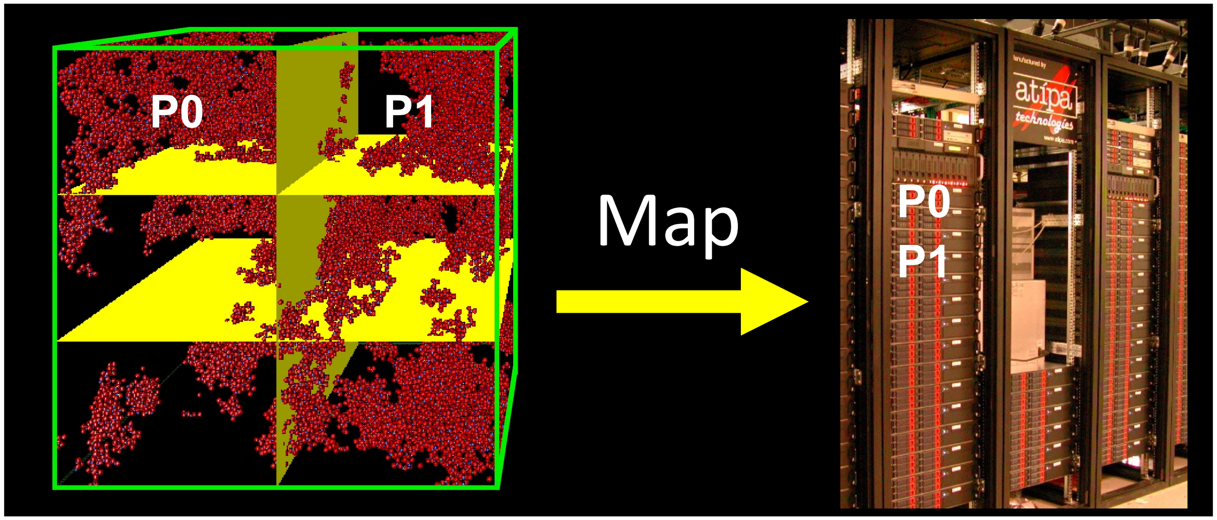
$$A^\alpha = (\phi/c\,, A^1, A^2, A^3)$$

$$\Box\, A^\alpha = \left(\frac{1}{c^2}\frac{\partial^2}{\partial t^2} - \nabla^2\right) A^\alpha = \frac{4\pi}{c} J^\alpha$$

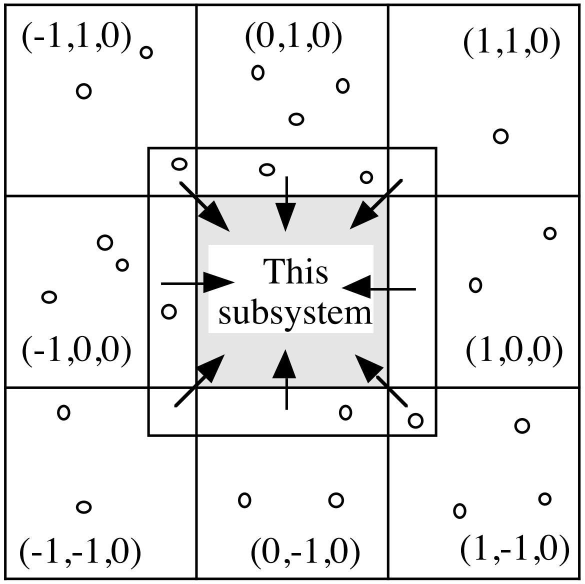
Parallel Molecular Dynamics

Spatial decomposition (short ranged): $O(N/P)$ computation

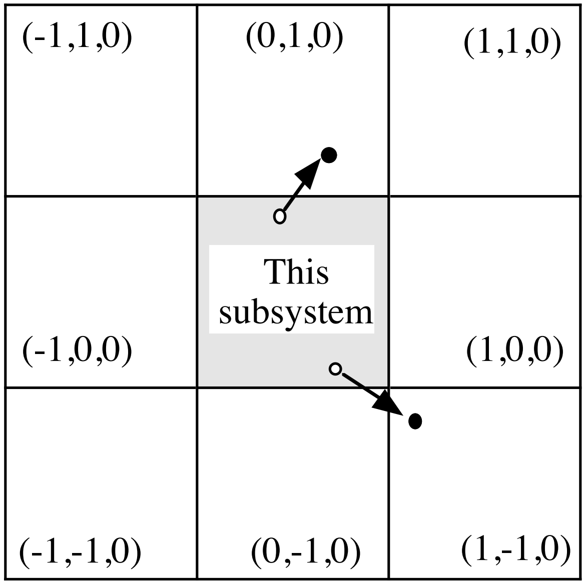




Atom caching: $O((N/P)^{2/3})$



Atom migration



See also parallel quantum dynamics lecture

History of Particle Simulations

’44 **John von Neumann memo on a stored-program computer:** ***“Our present analytical methods seem unsuitable for the solution of the important problems arising in connection with nonlinear partial differential equations. The really efficient high-speed computing devices may provide us with those heuristic hints which are needed in all parts of mathematics for genuine progress”***

’53 **First Monte Carlo simulation of liquid by Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller on MANIAC at Los Alamos Nat’l Lab**

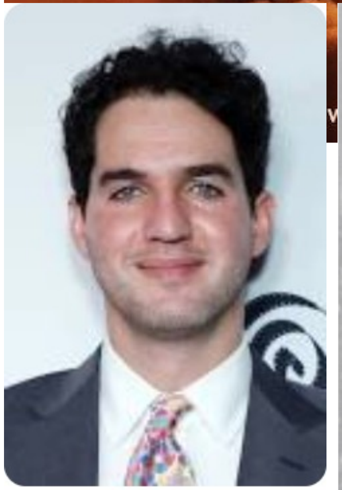
’55 **Enrico Fermi, John Pasta, and Stanislaw Ulam studied the dynamics of an one-dimensional array of particles coupled by anharmonic springs on MANIAC**

’56 **Dynamics of hard spheres (billiards) studied by Alder and Wainwright at Lawrence Livermore Nat’l Lab.**

’60 **Radiation damage in crystalline Cu studied with short-range repulsion and uniform attraction toward the center by George Vineyard’s group at Brookhaven Nat’l Lab**

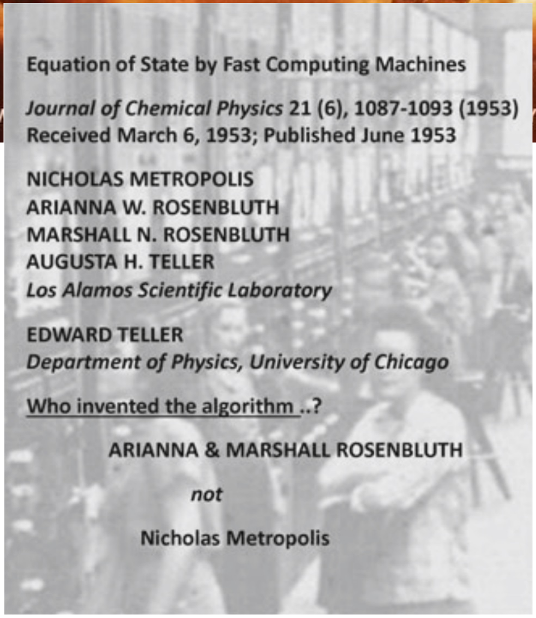
’64 **First MD simulation of liquid (864 argon atoms) using interatomic potentials by Aneesur Rahman at Argonne Nat’l Lab on a CDC 3600**

Simulation Pioneers in "Oppenheimer"



Benny Safdie

Edward Teller



Equation of State by Fast Computing Machines

Journal of Chemical Physics 21 (6), 1087-1093 (1953)
Received March 6, 1953; Published June 1953

NICHOLAS METROPOLIS
ARIANNA W. ROSENBLUTH
MARSHALL N. ROSENBLUTH
AUGUSTA H. TELLER
Los Alamos Scientific Laboratory

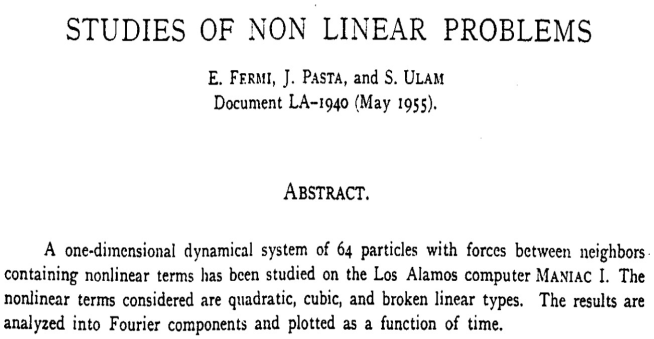
EDWARD TELLER
Department of Physics, University of Chicago

Who invented the algorithm ..?

ARIANNA & MARSHALL ROSENBLUTH

not

Nicholas Metropolis



STUDIES OF NON LINEAR PROBLEMS

E. Fermi, J. Pasta, and S. Ulam
Document LA-1940 (May 1955).

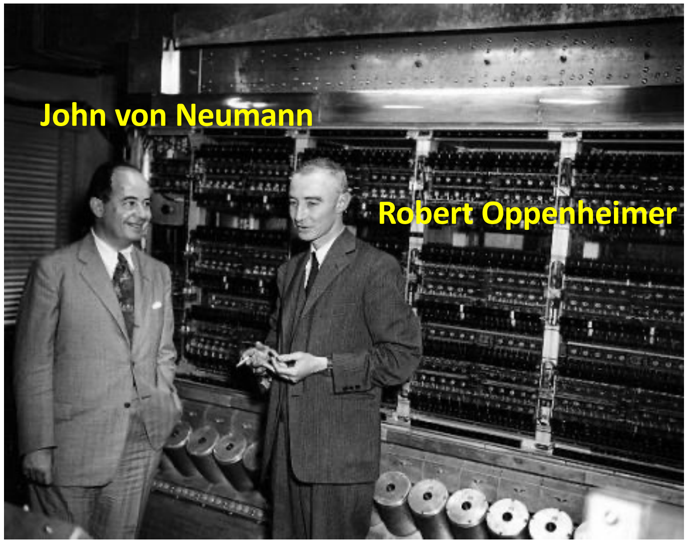
Abstract.

A one-dimensional dynamical system of 64 particles with forces between neighbors containing nonlinear terms has been studied on the Los Alamos computer Maniac I. The nonlinear terms considered are quadratic, cubic, and broken linear types. The results are analyzed into Fourier components and plotted as a function of time.

Danny Deferrari

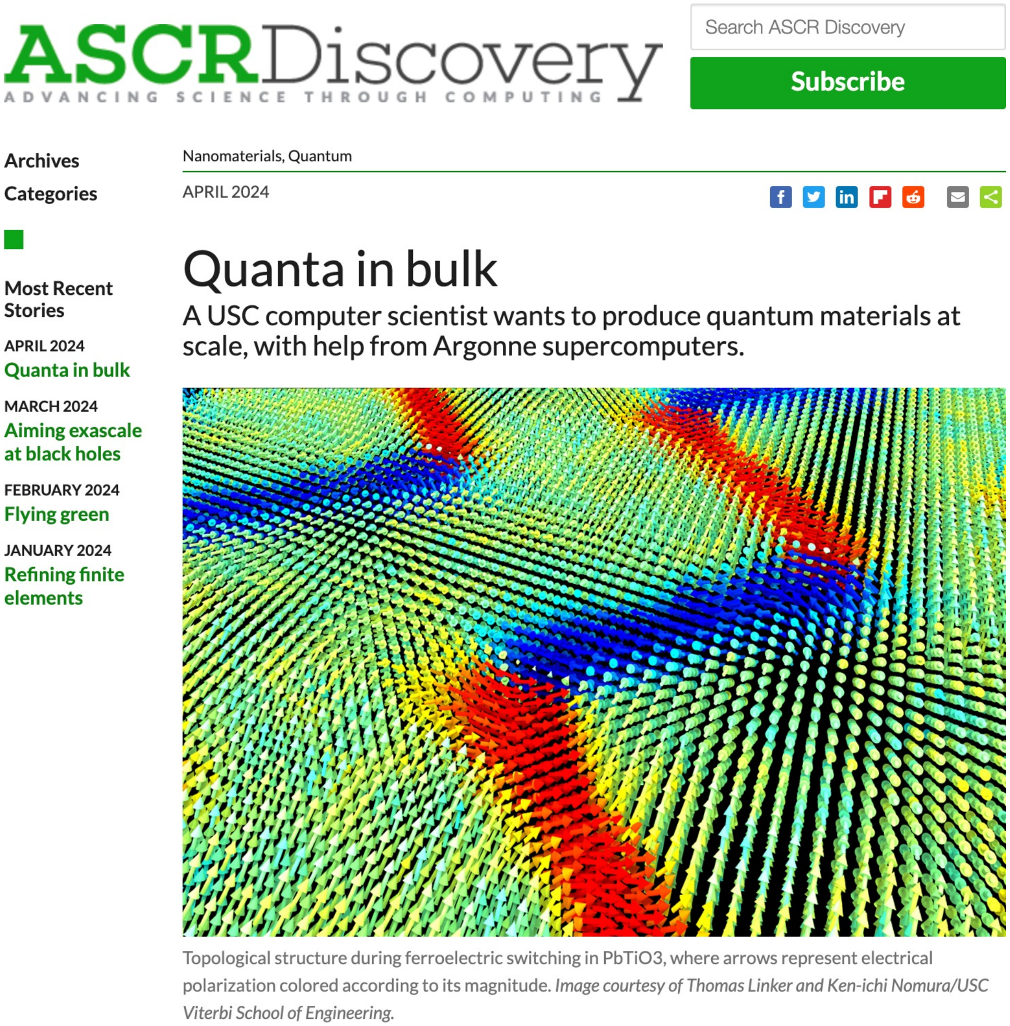
Dr Enrico Fermi

And US Supercomputers



MANIAC computer at Los Alamos

Aiichiro Nakano of the University of Southern California uses the Argonne Leadership Computing Facility to study how light can change the structures and properties of atomically thin materials to create better, cheaper semiconductors. (Image by Argonne National Laboratory.)



ASCRDiscovery

ADVANCING SCIENCE THROUGH COMPUTING

Search ASCR Discovery

Subscribe

Archives

Categories

Most Recent Stories

APRIL 2024
Quanta in bulk

MARCH 2024
Aiming exascale at black holes

FEBRUARY 2024
Flying green

JANUARY 2024
Refining finite elements

Nanomaterials, Quantum

APRIL 2024

Quanta in bulk

A USC computer scientist wants to produce quantum materials at scale, with help from Argonne supercomputers.

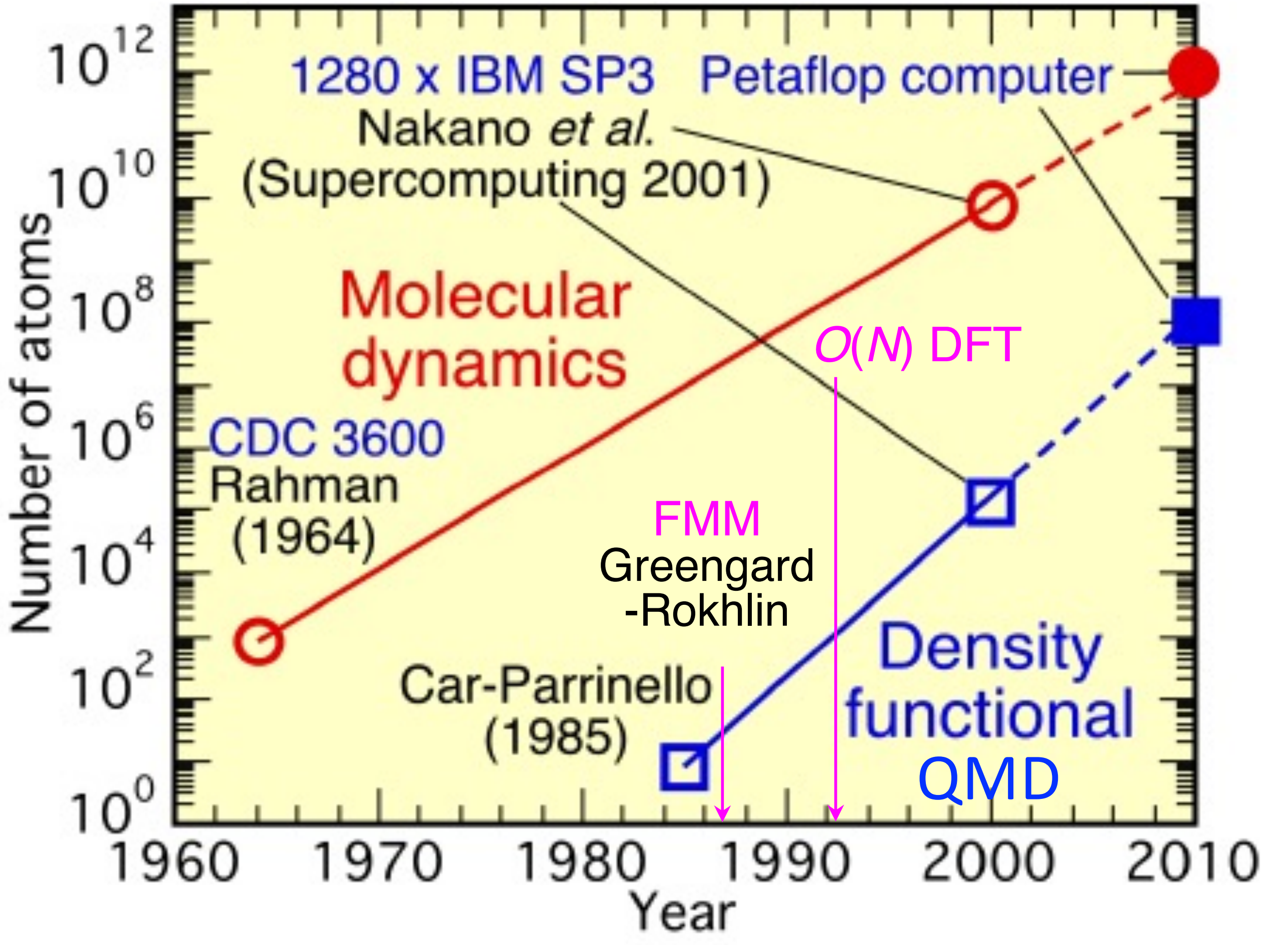
Topological structure during ferroelectric switching in PbTiO3, where arrows represent electrical polarization colored according to its magnitude. *Image courtesy of Thomas Linker and Ken-ichi Nomura/USC Viterbi School of Engineering.*

Supercomputing at Argonne National Laboratory now

Moore's Law in Scientific Computing

Number of particles in MD simulations has doubled:

- **Every 19 months in 50 years for classical MD**
- **Every 22 months in 30 years for quantum MD (QMD)**

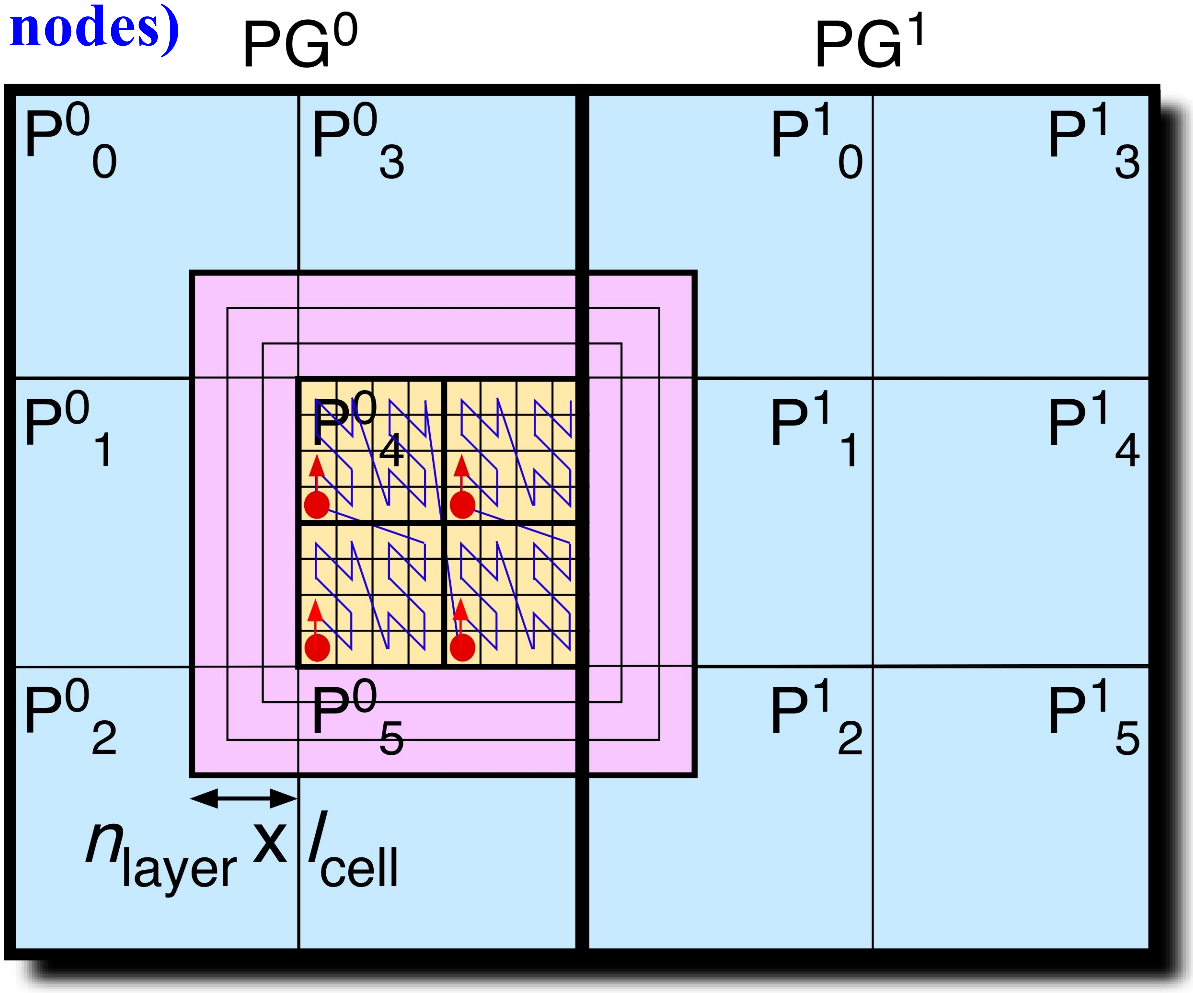


2014: 10^{12}-atom MD & 10^{8}-electron QMD on a 10 petaflop/s Blue Gene/Q with advances in algorithmic & parallel-computing techniques

Tunable Hierarchical Cellular Decomposition

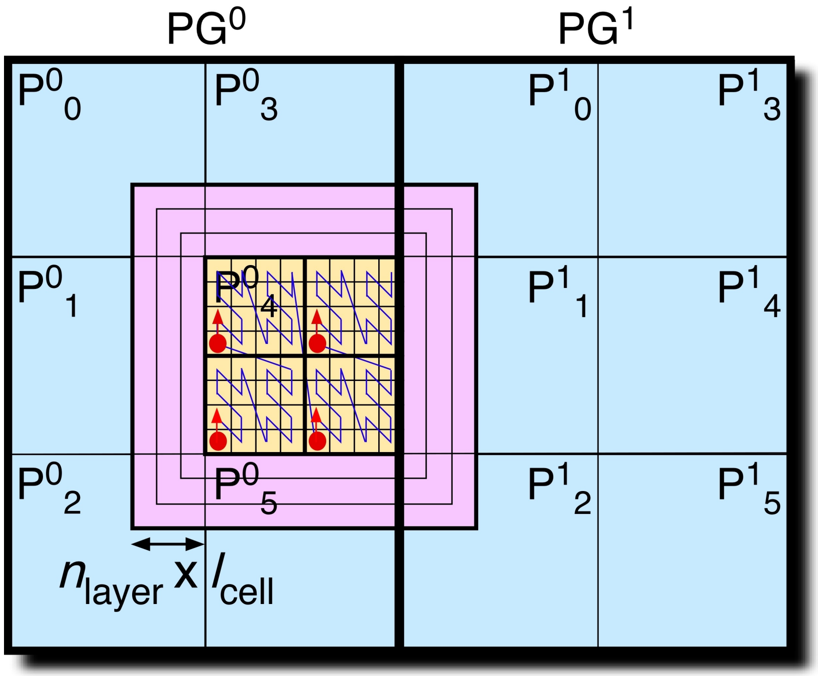
Mapping $O(N)$ divide-&-conquer algorithms onto memory hierarchies

- Spatial decomposition with data “caching” & “migration”
- Computational cells (*e.g*., linked-list cells in MD) < cell blocks (threads) < processes (P^{γ}_{π}, spatial decomposition subsystems) < process groups (P^{γ}, Grid nodes)
- Multilayer cellular decomposition (MCD) for *n*-tuples (*n* = 2–6)
- Tunable cell data & computation structures: Data/computation re-ordering & granularity parameterized at each decomposition level
- Tunable hybrid MPI + OpenMP + SIMD implementation



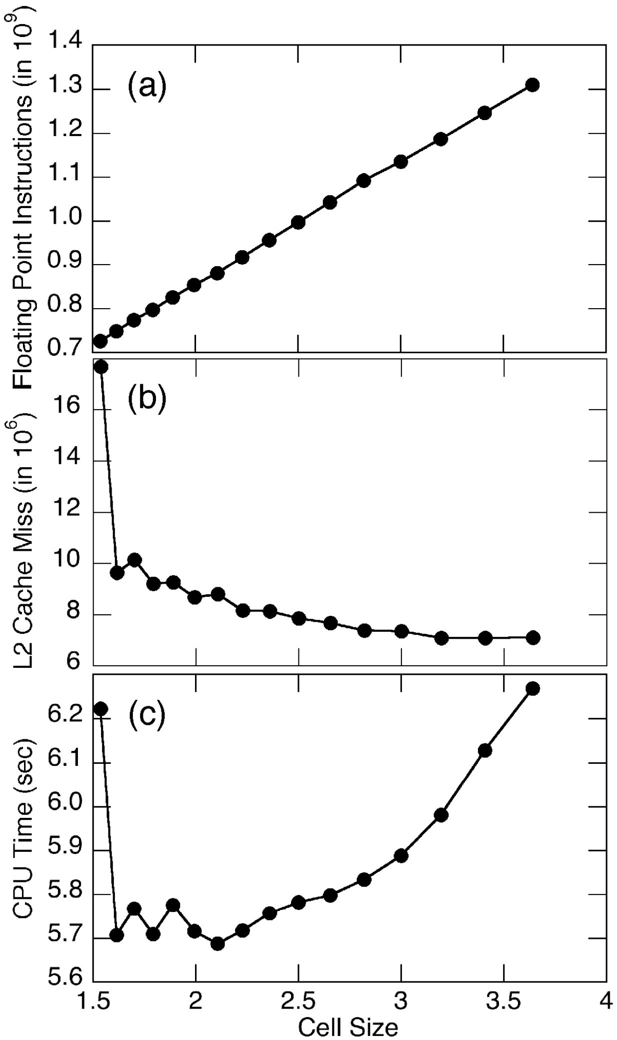
Nomura *et al.*, *IPDPS 2009*

Performance Tunability



Floating-point operation/L2 cache miss trade-off:

331,776-atom silica MRMD on 1.4GHz Pentium III



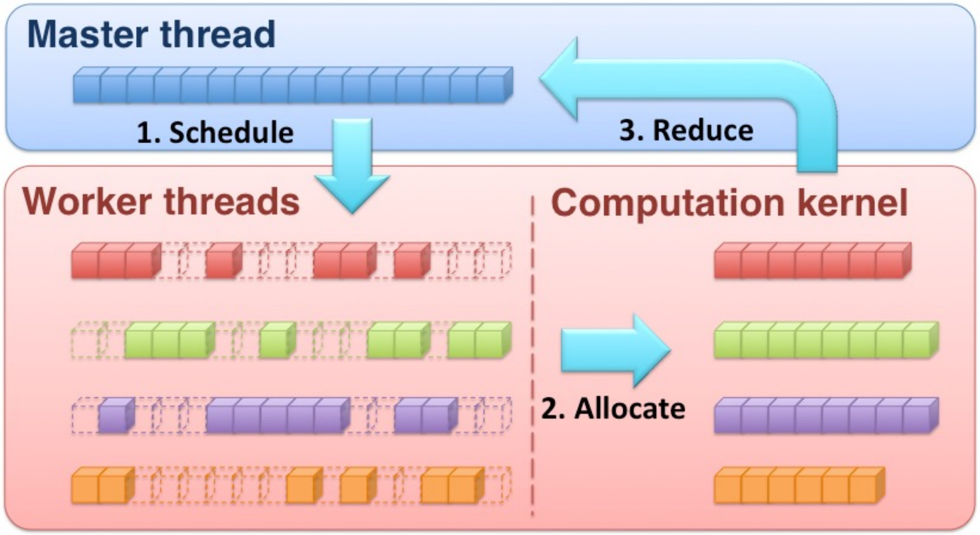
MPI/OpenMP parallelism trade-off:
8,232,000-atom silica MRMD & 290,304-atom RDX F-ReaxFF on 8-way 1.5 GHz Power4

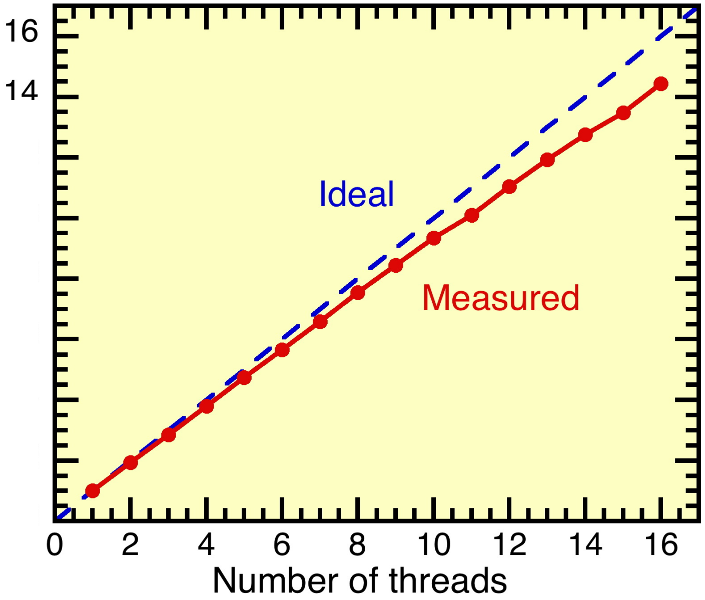
Number of OpenMP threads, n_{td}	Number of MPI processes, n_p	Execution time/MD time step (sec)	
		MRMD	P-ReaxFF
1	8	4.19	62.5
2	4	5.75	58.9
4	2	8.60	54.9
8	1	12.5	120

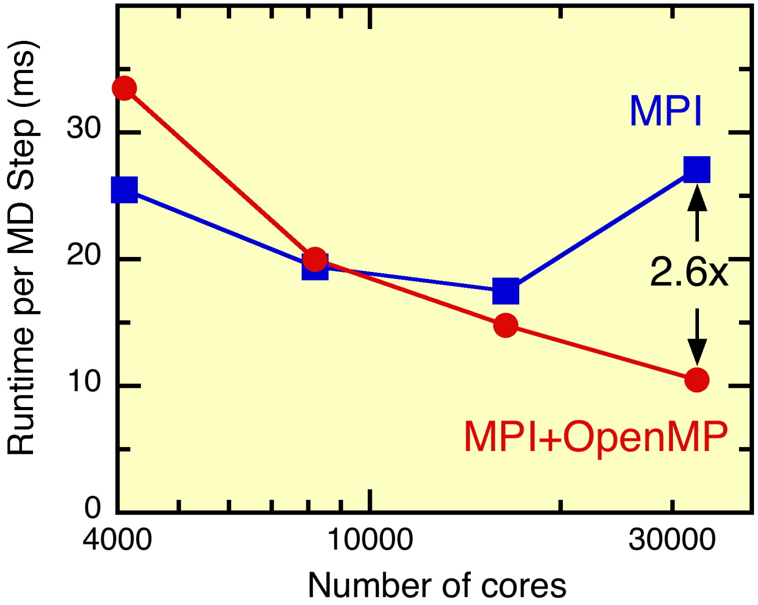
Spatially Compact Thread Scheduling

Concurrency-control mechanism: Data privatization # of atoms

- **Reduced memory:** # of threads
  $\Theta(nq) \rightarrow \Theta(n+n^{2/3}q^{1/3})$
- **Strong scaling parallel efficiency 0.9 on quad quad-core AMD Opteron**
- **2.6× speedup over MPI by hybrid MPI+OpenMP on 32,768 IBM Blue Gene/P cores**







M. Kunaseth *et al.*, *PDPTA'11*; *J. Supercomput*. ('13)

Concurrency-Control Mechanisms

A number of concurrency-control mechanisms (CCMs) are provided by OpenMP to coordinate multiple threads:

- Critical section: Serialization
- Atomic update: Expensive hardware instruction
- Data privatization: Requires large memory $\Theta(nq)$
- Hardware transactional memory: Rollbacks (on IBM Blue Gene/Q)

# of atoms per node

# of threads

CCM performance varies:

- Depending on computational characteristics of each program
- In many cases, CCM degrades performance significantly

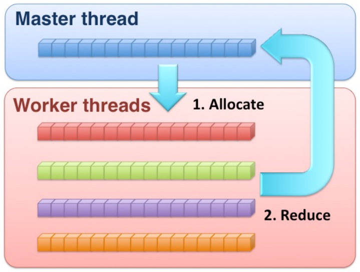
HTM/critical section

```
#pragma omp <critical|tm_atomic>
{
  ra[i][0] += fa*dr[0];
  ra[i][1] += fa*dr[1];
  ra[i][2] += fa*dr[2];
}
```

Atomic update

```
#pragma omp atomic
ra[i][0] += fa*dr[0];
#pragma omp atomic
ra[i][1] += fa*dr[1];
#pragma omp atomic
ra[i][2] += fa*dr[2];
```

Data privatization



Goal: Provide a guideline to choose the “right” CCM

Hardware Transactional Memory

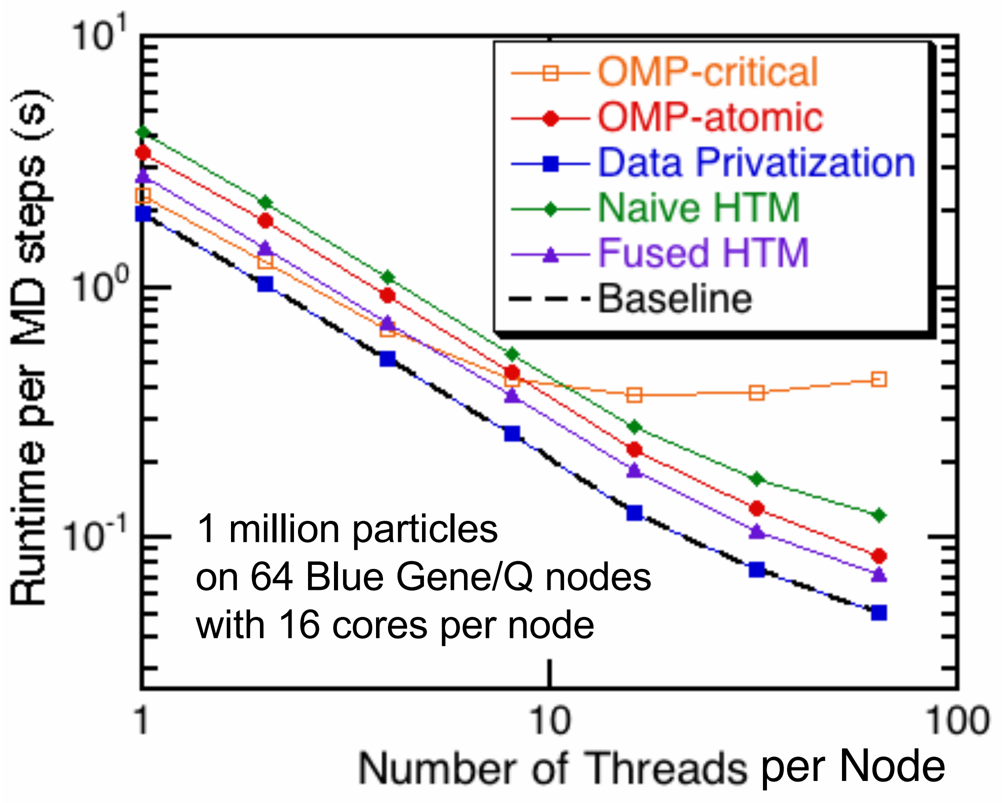
Transactional memory (TM): An opportunistic CCM

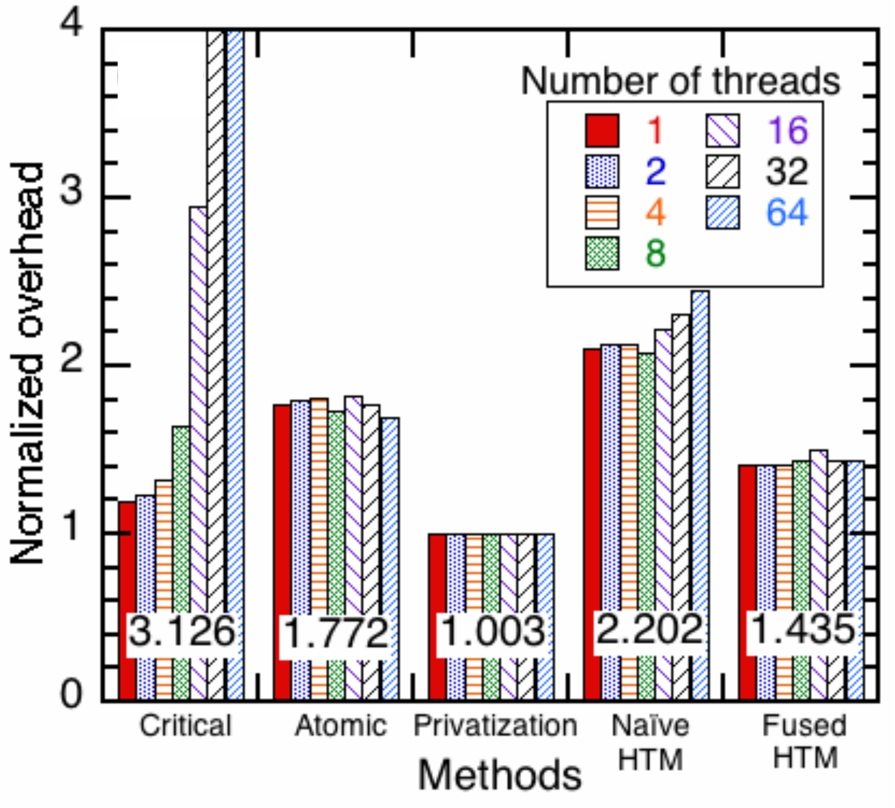
- **Avoids memory conflicts by monitoring a set of speculative operations (*i.e.* transaction)**
- **If two or more transactions write to the same memory address, transaction(s) will be restarted—a process called rollback**
- **If no conflict detected in the end of a transaction, operations within the transaction becomes permanent (*i.e.* committed)**
- **Software TM usually suffers from large overhead**

Hardware TM on IBM Blue Gene/Q:

- **The first commercial platform implementing TM support at hardware level *via* multiversioned L2-cache**
- **Hardware support is expected to reduce TM overhead**
- **Performance of HTM on molecular dynamics has not been quantified**

Strong-Scaling Benchmark for MD





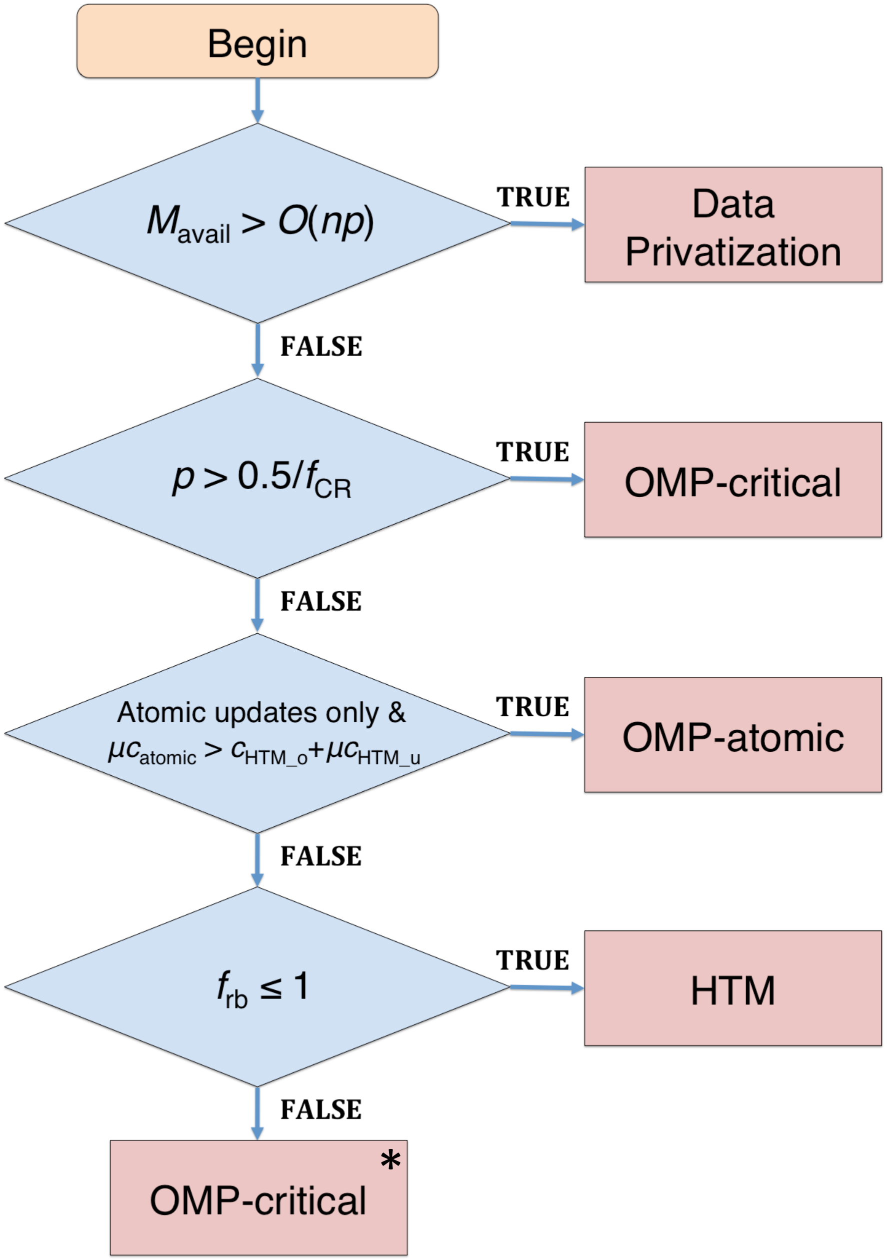
*Baseline: No CCM; the result is wrong

Developed a fundamental understanding of CCMs:

- **OMP-critical has limited scalability on larger number of threads (q > 8)**
- **Data privatization is the fastest, but it requires $\Theta(nq)$ memory**
- **Fused HTM performs the best among constant-memory CCMs**

M. Kunaseth *et al.*, *PDSEC'13*

Threading Guideline for Scientific Programs



Begin

$M_{\mathrm{avail}} > O(np)$

TRUE → Data Privatization

FALSE

$p > 0.5/f_{\mathrm{CR}}$

TRUE → OMP-critical

FALSE

Atomic updates only & $\mu c_{\mathrm{atomic}} > c_{\mathrm{HTM\_o}} + \mu c_{\mathrm{HTM\_u}}$

TRUE → OMP-atomic

FALSE

$f_{\mathrm{rb}} \leq 1$

TRUE → HTM

FALSE

OMP-critical *

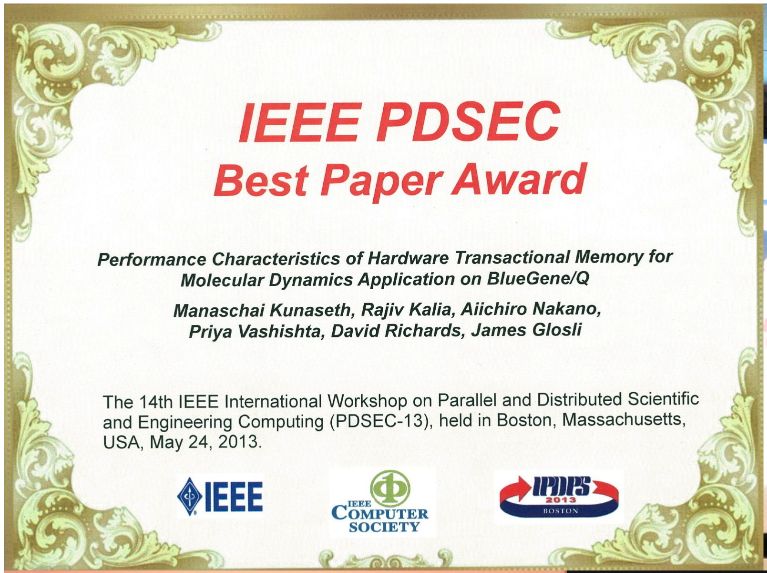
M. Kunaseth *et al.*, *PDSEC'13*

Focus on minimizing runtime (best performance):

- **Have enough memory → data privatization**
- **Conflict region is small → OMP-critical**
- **Small amount of updates → OMP-atomic**
- **Conflict rate is low → HTM**
- **Other → OMP-critical* (poor performance)**

Concurrency control mechanism	Parallel efficiency
OMP-critical	$e = \min(\frac{1}{pf_{\mathrm{CR}}}, 1)$
OMP-atomic	$e = \frac{t_{\mathrm{total}}}{t_{\mathrm{total}} + m\mu c_{\mathrm{atomic}}}$
Data privatization	$e = \frac{t_{\mathrm{total}}}{t_{\mathrm{total}} + c_{\mathrm{reduction}} n \log p}$
HTM	$e = \frac{t_{\mathrm{total}}}{t_{\mathrm{total}} + m(c_{\mathrm{HTM\_overhead}} + \mu c_{\mathrm{HTM\_update}})}$

IEEE PDSEC Best Paper & Beyond

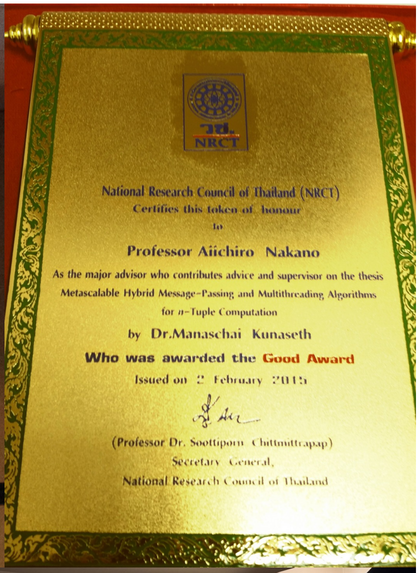


IEEE PDSEC
Best Paper Award

Performance Characteristics of Hardware Transactional Memory for Molecular Dynamics Application on BlueGene/Q

Manaschai Kunaseth, Rajiv Kalia, Aiichiro Nakano, Priya Vashishta, David Richards, James Glosli

The 14th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC-13), held in Boston, Massachusetts, USA, May 24, 2013.



National Research Council of Thailand (NRCT)
Certifies this token of honour
to

Professor Aiichiro Nakano

As the major advisor who contributes advice and supervisor on the thesis
Metascalable Hybrid Message-Passing and Multithreading Algorithms
for *n*-Tuple Computation

by Dr.Manaschai Kunaseth

Who was awarded the Good Award

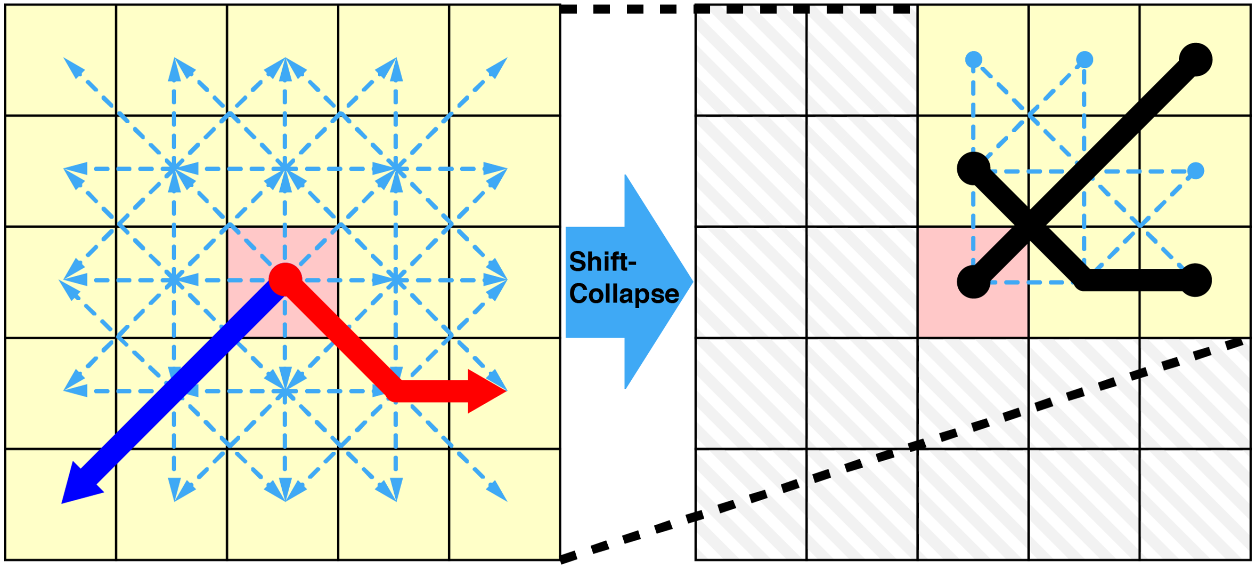
Issued on 2 February 2015

(Professor Dr. Soottiporn Chittmittrapap)
Secretary General,
National Research Council of Thailand

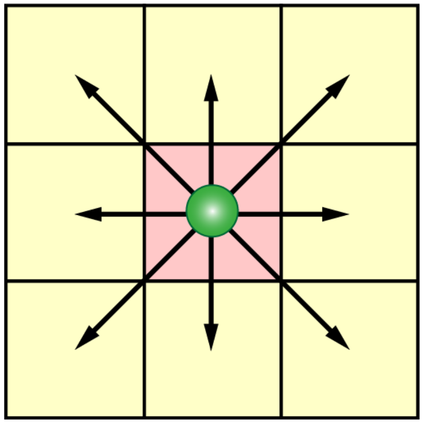
THE 21st INTERNATIONAL ANNUAL SYMPOSIUM ON
COMPUTATIONAL SCIE AND ENGINEERING
AUGUST 2-5 2017
THAILAND SCIENCE PARK ANI, THAILAND
HDDC

Shift-Collapse (SC) Algorithm

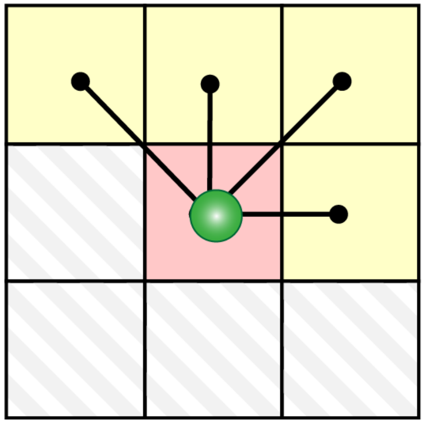
- **Generalization of Shaw's eighth-cell method (non-owner-compute method on high-latency cluster) for pair computation to general dynamic range-limited *n*-tuples**



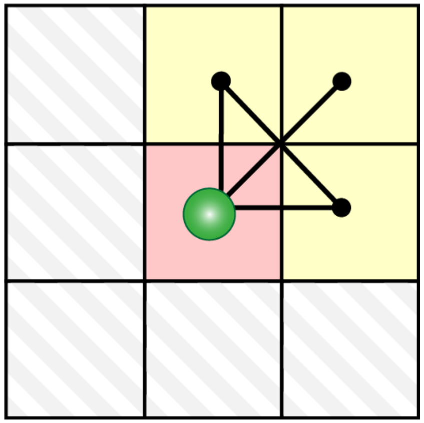
M. Kunaseth *et al*., *IEEE/ACM Supercomputing (SC13)*



Full-shell (FS) method
e.g., Rappaport, '88



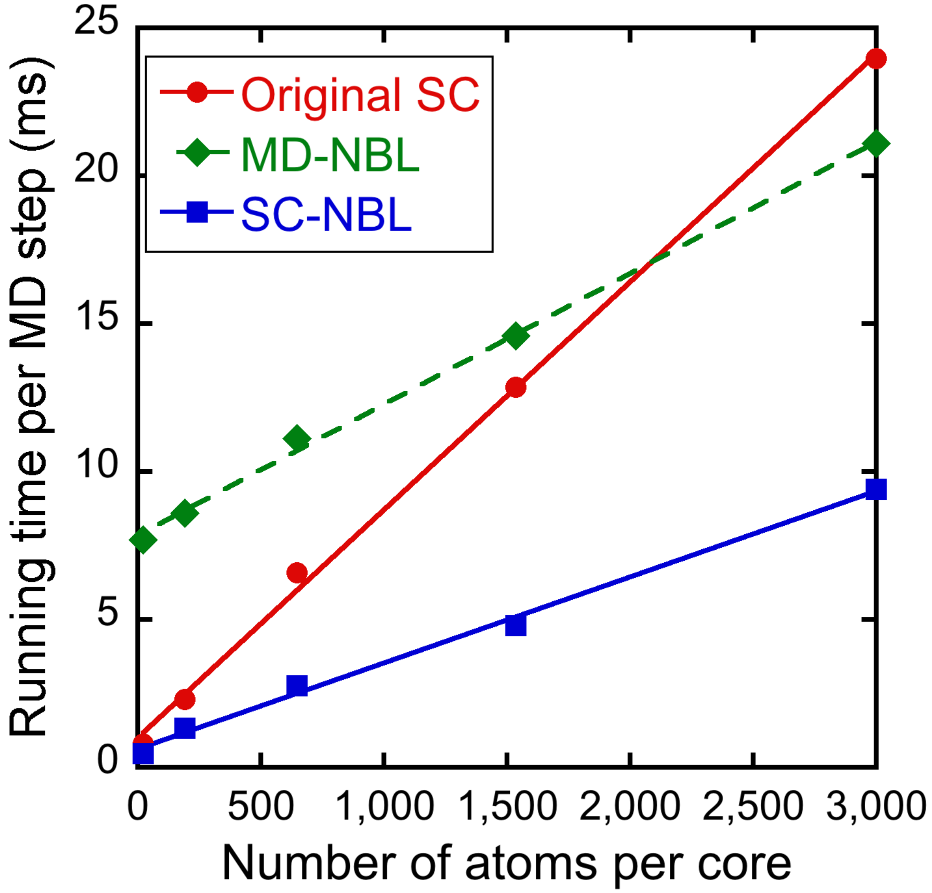
Half-shell (HS) method
e.g., Rappaport, '88



Eighth-shell (ES) method
Bower *et al*., '06

Shift-Collapse on Neighbor List (SC-NBL)

- **Apply shift-collapse operations to the hybrid linked-cell & neighbor list code (best of both)**



Shift/collapse on neighbor list (SC-NBL): fast evaluation of dynamic many-body potentials in molecular dynamics simulations, M. Kunaseth, S. Hannongbua, & A. Nakano, *Comput. Phys. Commun.* 235, 88 (2019)

Challenge: Expose massive data parallelism for SC on graphics processing unit (GPU)

GPU Programming: CUDA

- Compute Unified Device Architecture
- Integrated host (CPU) + device (GPU) application programming interface based on C language developed at NVIDIA
- CUDA homepage

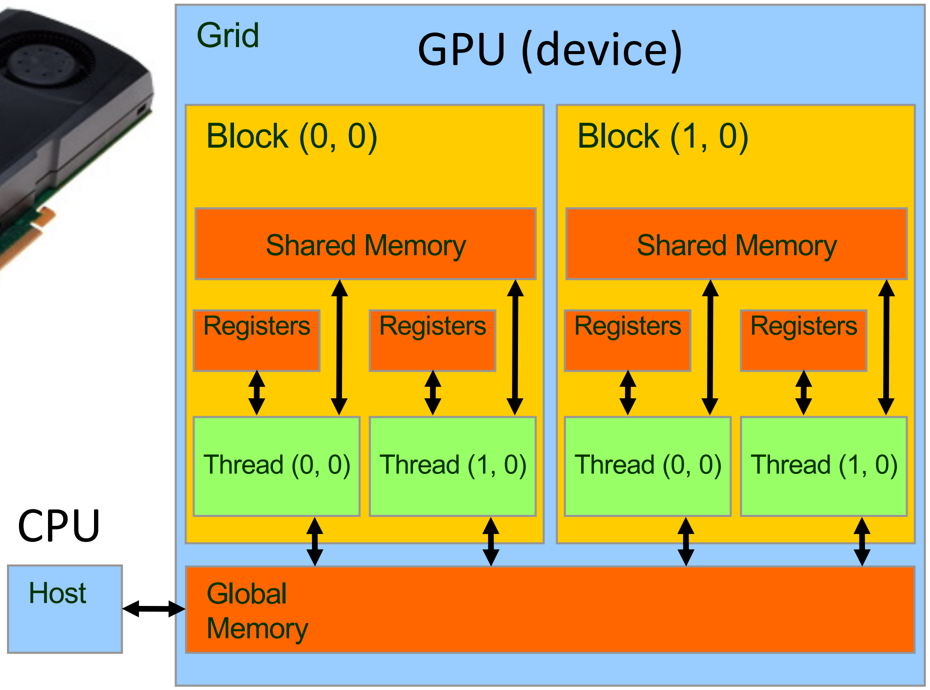
```
http://www.nvidia.com/object/cuda_home_new.html
```

- Compilation

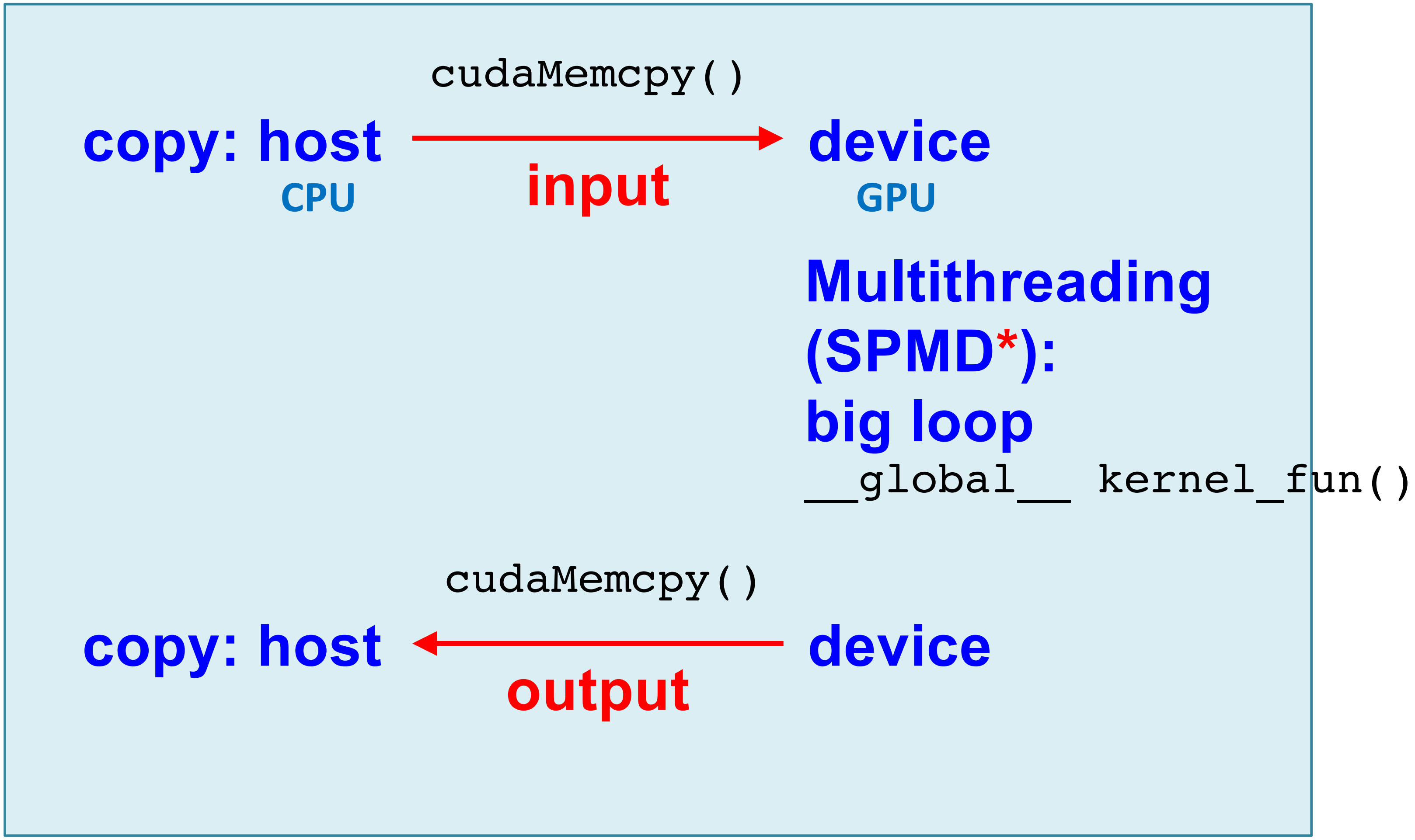
```
$ nvcc pi.cu
```

- Execution

```
$ a.out
PI = 3.141593
```



Summary: CUDA Computing



cudaMemcpy()

copy: host → device

CPU input GPU

Multithreading (SPMD*): big loop

`__global__ kernel_fun()`

cudaMemcpy()

copy: host ← device

output

* Single program multiple data we have learned is called single instruction multiple threads (SIMT) in GPU terminology

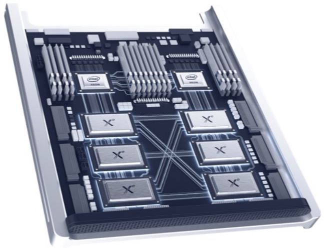
See https://aiichironakano.github.io/cs596/CUDA.pdf

Exaflop/s Supercomputing

- **Diverse exaflop/s supercomputing platforms**

Summit (0.2 Exaflop/s)
IBM CPU/NVIDIA GPU

1 HPC and AI Optimized
AMD EPYC CPU
4 Purpose Built AMD
Radeon Instinct GPU

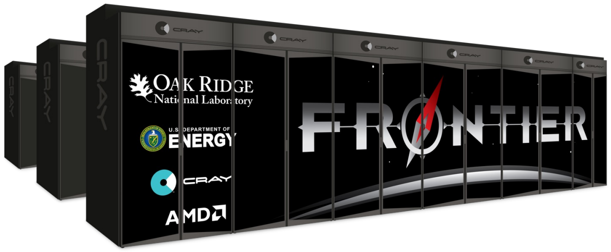


GPU Architecture

X^e arch-based "Ponte Vecchio" GPUTile-based, chiplets, HBM stack, Foveros 3D integration, 7nm

On-Node Interconnect

CPU-GPU: PCIe
GPU-GPU: X^e Link



Frontier (1.7 Exaflop/s)
AMD CPU/AMD GPU

Aurora (2.0 Exaflop/s)
Intel CPU/Intel GPU

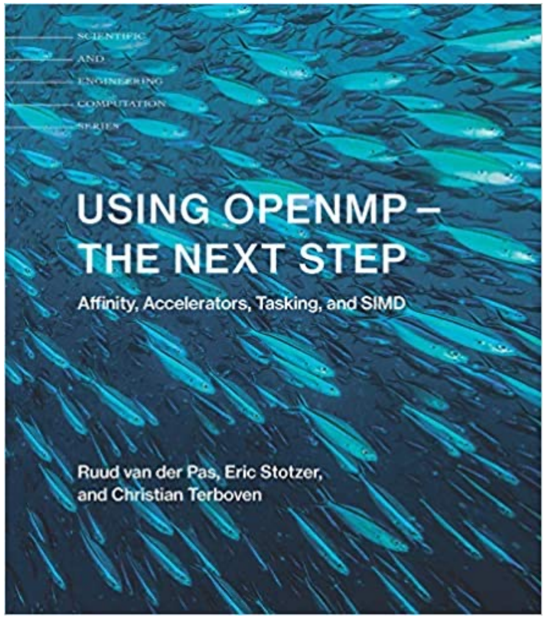
- **Need an *open* programming model for *heterogeneous* (*e.g.*, GPU-accelerated) clusters (note CUDA is a proprietary language by NVIDIA)**

See https://extremecomputingtraining.anl.gov/agenda-2024/

Open Programming Models: Easy Way

- **OpenACC (Open Computing Language)**
  Open standard for directive-based programming of heterogeneous devices

  https://www.openacc.org/



- **OpenMP 4.5/5**
  Starting specification version 4.5, OpenMP allows offloading the execution of code & data to heterogeneous devices

  https://www.openmp.org/specifications/

OpenMP Target Offload

- **Simple example**

```
main() {
  float a[1000],b[1000],c,d;
  ...
  #pragma omp target map(a,b,c,d)
  {
    int i;
    #pragma omp parallel for
    for (i=0; i<N; i++)
      a[i] = b[i]*c+d;
  }
  ...
}
```

See https://aiichironakano.github.io/cs596/OMPtarget.pdf

- **When a host thread encounter the `#pragma omp target` directive, the target region specified by it will be executed by a new thread running on an accelerator,** *cf.* CUDA GPU kernel
- **Before the new thread starts executing the target region, the variable in the `map()` clause are mapped onto accelerator memory, which often is disjunct from the host memory,** *cf.* `cudaMemcpy()`
- **The offloaded code is usually a data-parallel structured block, which can be handled by multiple threads on accelerator using standard OpenMP constructs like `#pragma omp parallel for`**

Open Programming Models: Hard Way

- **OpenCL (Open Computing Language)**

  Open standard for programming heterogeneous devices

  https://www.khronos.org/opencl/

- **SYCL**

  High-level programming standard (or abstraction layer) for single-source C++ based language on heterogeneous computer architectures

  https://www.khronos.org/sycl/

- **Data parallel C++ (DPC++)**

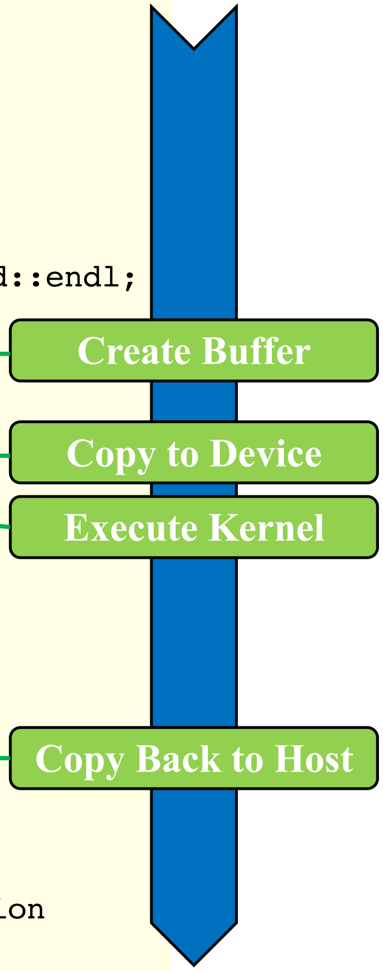
  Extension of C++ programming language, incorporating SYCL & other features, initially created by Intel; an open-source compiler is available on GitHub

  https://intel.github.io/llvm-docs/index.html

See https://aiichironakano.github.io/cs596/SYCL.pdf

SYCL Program Pattern

```
#include <CL/sycl.hpp>
#include <iostream>
#include <array>
using namespace cl::sycl;
#define NBIN 1000000  // # of bins for quadrature
#define NTRD 512      // # of threads
int main() {
  float step = 1.0f/NBIN;
  std::array<float, NTRD> sum;
  for (int i=0; i<NTRD; ++i) sum[i] = 0.0f;
  {
    sycl::queue q(sycl::gpu_selector_v);
    std::cout << "Running on: " <<
      q.get_device().get_info<info::device::name>() << std::endl;
    range<1> sizeBuf{NTRD};
    buffer<float, 1> sumBuf(sum.data(), sizeBuf);
    q.submit([&](handler &h){
      auto sumAccessor =
      sumBuf.get_access<access::mode::read_write>(h);
      h.parallel_for(sizeBuf, [=](id<1> tid) {
        for (int i=tid; i<NBIN; i+=NTRD) {
          float x = (i+0.5f)*step;
          sumAccessor[tid] += 4.0f/(1.0f+x*x);
        }
      });  // End parallel for
    });  // End queue submit
  }
  float pi=0.0f;
  for (int i=0; i<NTRD; i++)  // Thread reduction
    pi += sum[i];
  pi *= step;  // Multiply bin width to complete integration
  std::cout << "Pi = " << pi << std::endl;
  return 0;
}
```


Create Buffer

Copy to Device

Execute Kernel

Copy Back to Host

GPU: Easy & Hard Ways

Serial: pi.c

```
#include <stdio.h>
#define NBIN 1000000000
int main() {
   double step, x, sum=0.0, pi;
   step = 1.0/NBIN;
   for (long long i=0; i<NBIN; i++) {
      x = (i+0.5)*step;
      sum += 4.0/(1.0+x*x);
   }
   pi = sum*step;
   printf("PI = %f\n",pi);
   return 0;
}
```

≲

OpenMP: omp_target_pi.c

```
#include <omp.h>
#include <stdio.h>
#define NBIN 1000000
int main() {
   float step,sum=0.0,pi;
   step = 1.0/(float)NBIN;
   #pragma omp target map(step,sum)
   {
      # pragma omp parallel for reduction(+:sum)
      for (long long i=0; i<NBIN; i++) {
         float x = (i+0.5)*step;
         sum += 4.0/(1.0+x*x);
      }
   }
   pi = sum*step;
   printf("PI = %f\n",pi);
   return 0;
}
```

https://aiichironakano.github.io/cs596-code.html

≪

SYCL: pi.cpp

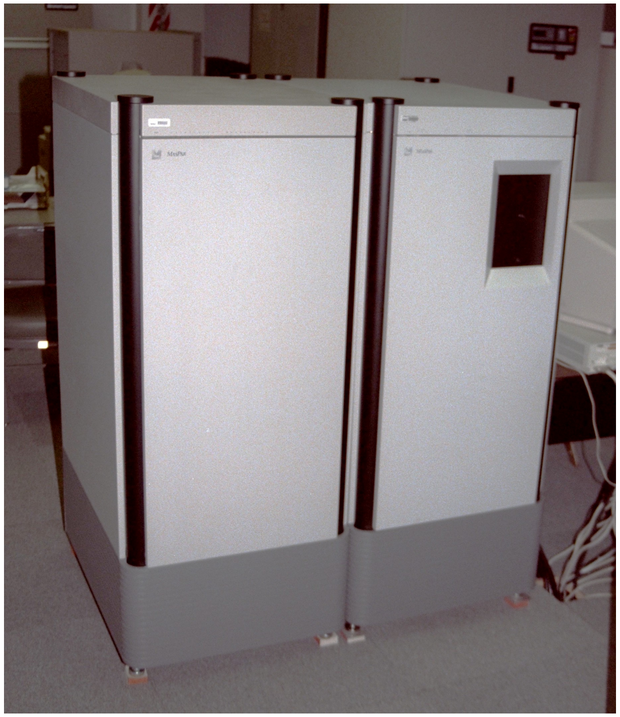
```
#include <CL/sycl.hpp>
#include <iostream>
#include <array>

using namespace cl::sycl;

#define NBIN 1000000   // # of bins for quadrature
#define NTRD 512       // # of threads

int main()
{
  float step = 1.0f/NBIN;
  std::array<float, NTRD> sum;
  for (int i=0; i<NTRD; ++i) sum[i] = 0.0f;

  queue q(gpu_selector{});

  std::cout << "Running on: " <<
  q.get_device().get_info<info::device::name>() << std::endl;

  range<1> sizeBuf{NTRD};

  {
    buffer<float, 1> sumBuf(sum.data(), sizeBuf);
    q.submit([&](handler &h){
      auto sumAccessor =
      sumBuf.get_access<access::mode::read_write>(h);
      h.parallel_for(sizeBuf, [=](id<1> tid) {
        for (int i=tid; i<NBIN; i+=NTRD) {
          float x = (i+0.5f)*step;
          sumAccessor[tid] += 4.0f/(1.0f+x*x);
        }
      });  // End parallel for
    });  // End queue submit
  }

  float pi=0.0f;
  for (int i=0; i<NTRD; i++)  // Inter-thread reduction
    pi += sum[i];
  pi *= step;  // Multiply bin width to complete integration

  std::cout << "Pi = " << pi << std::endl;

  return 0;
}
```

≲

CUDA: pi.cu

```
// Using CUDA device to calculate pi
#include <stdio.h>
#include <cuda.h>

#define NBIN  10000000  // Number of bins
#define NUM_BLOCK   13  // Number of thread blocks
#define NUM_THREAD 192  // Number of threads per block
int tid;
float pi = 0;

// Kernel that executes on the CUDA device
__global__ void cal_pi(float *sum, int nbin, float step, int nthreads, int nblocks) {
   int i;
   float x;
   int idx = blockIdx.x*blockDim.x+threadIdx.x;  // Sequential thread index across the blocks
   for (i=idx; i< nbin; i+=nthreads*nblocks) {
      x = (i+0.5)*step;
      sum[idx] += 4.0/(1.0+x*x);
   }
}

// Main routine that executes on the host
int main(void) {
   dim3 dimGrid(NUM_BLOCK,1,1);  // Grid dimensions
   dim3 dimBlock(NUM_THREAD,1,1);  // Block dimensions
   float *sumHost, *sumDev;  // Pointer to host & device arrays

   float step = 1.0/NBIN;  // Step size
   size_t size = NUM_BLOCK*NUM_THREAD*sizeof(float);  //Array memory size
   sumHost = (float *)malloc(size);  //  Allocate array on host
   cudaMalloc((void **) &sumDev, size);  // Allocate array on device
   // Initialize array in device to 0
   cudaMemset(sumDev, 0, size);
   // Do calculation on device
   cal_pi <<<dimGrid, dimBlock>>> (sumDev, NBIN, step, NUM_THREAD, NUM_BLOCK); // call CUDA kernel
   // Retrieve result from device and store it in host array
   cudaMemcpy(sumHost, sumDev, size, cudaMemcpyDeviceToHost);
   for(tid=0; tid<NUM_THREAD*NUM_BLOCK; tid++)
      pi += sumHost[tid];
   pi *= step;

   // Print results
   printf("PI = %f\n",pi);

   // Cleanup
   free(sumHost);
   cudaFree(sumDev);

   return 0;
}
```

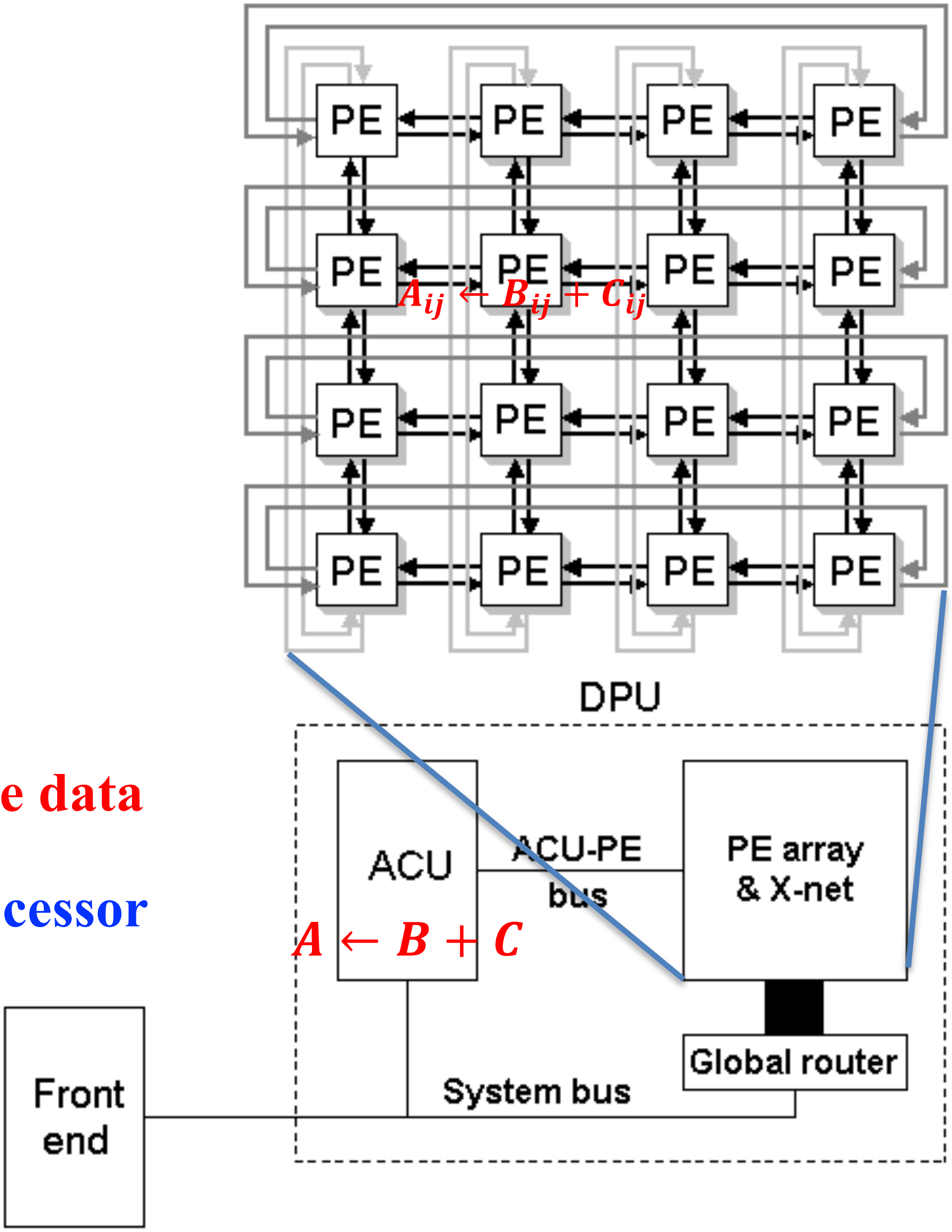
Massive SIMD Data-Parallel Accelerator



SIMD: single-instruction multiple data

Quantum dynamics on 8,192-processor (128 × 64) MasPar 1208B

Nakano,
Comput. Phys. Commun.
83, 181 ('94)



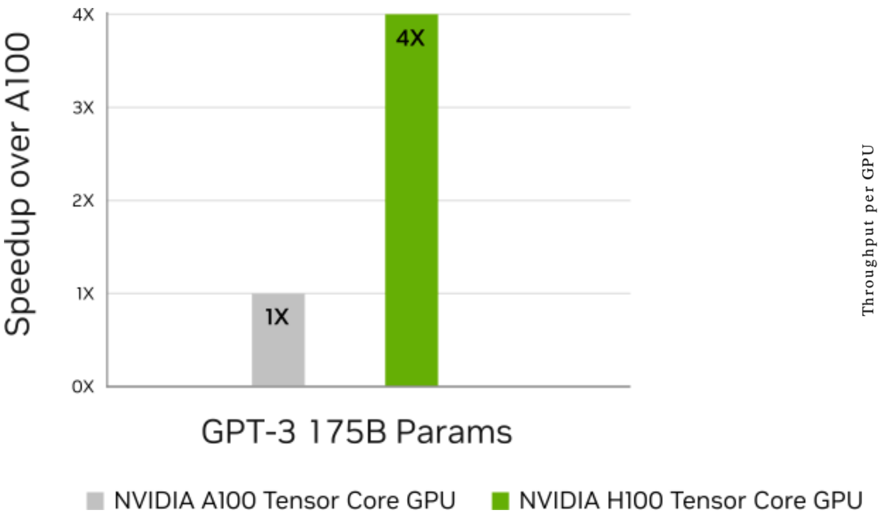
$A_{ij} \leftarrow B_{ij} + C_{ij}$

$A \leftarrow B + C$

See lecture on pre-Beowulf parallel computing

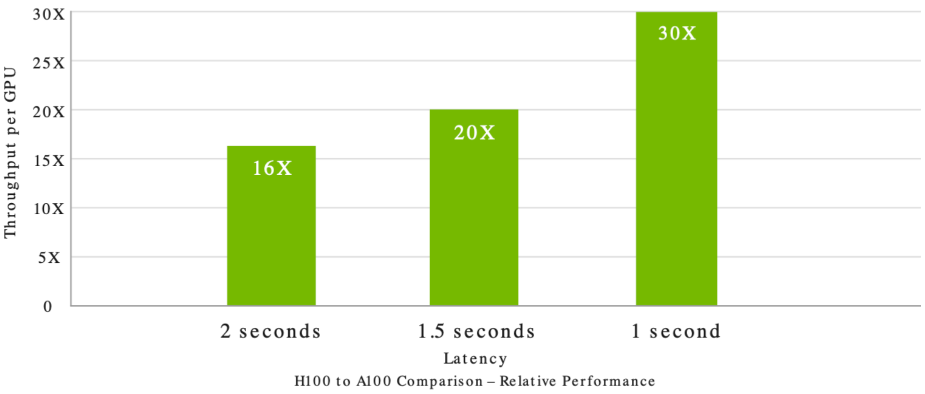
New Generation of GPU

- **H100 is here: 18,432 CUDA cores & 640 tensor cores**

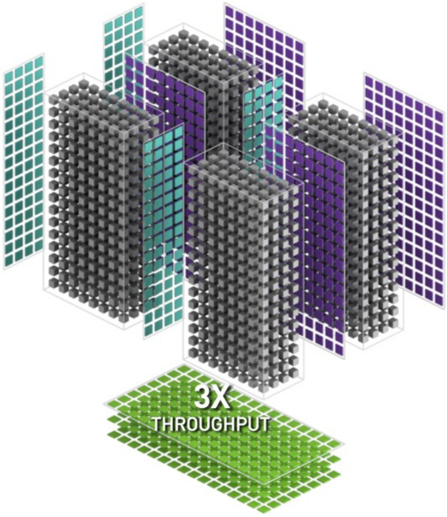


Up to 30X higher AI inference performance on the largest models

Megatron chatbot inference (530 billion parameters)



H100 FP16

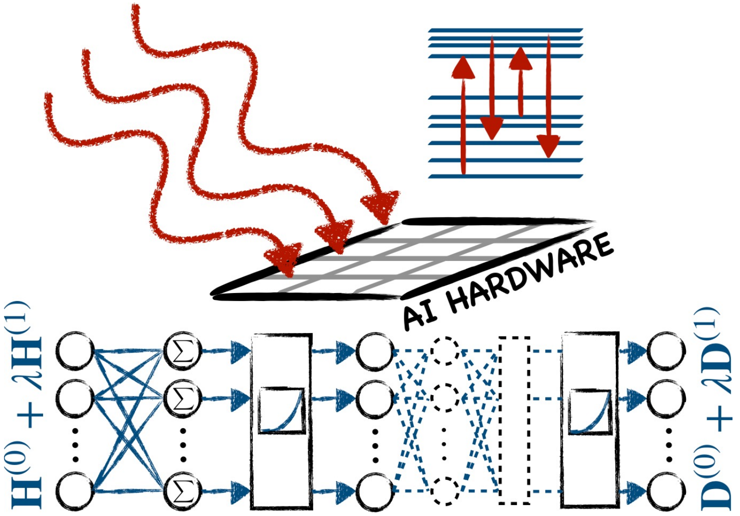


- **Unlike general-purpose CUDA cores, tensor cores are specialized processing units designed for (mixed-precision) matrix operations in deep learning**

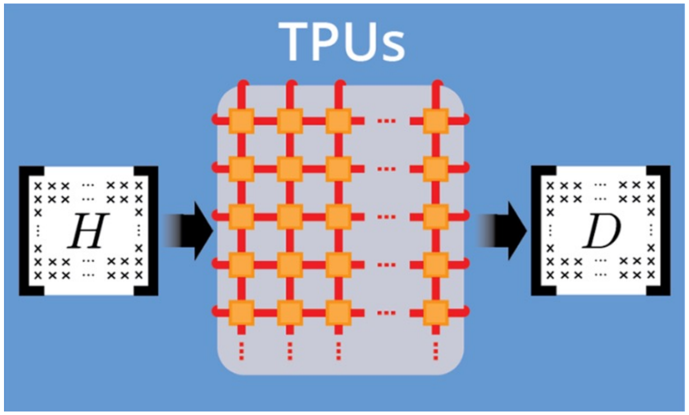
Scientific Tensor Computing?

NVIDIA tensor cores to Google tensor processing unit (TPU) & beyond

- **Joshua Finkelstein *et al.*, "Quantum perturbation theory using tensor cores and a deep neural network,"** *J. Chem. Theo. Comput.* 18, 4255 ('22)



- **Ryan Pederson *et al.*, "Large scale quantum chemistry with tensor processing units,"** *J. Chem. Theo. Comput.* 19, 25 ('23)



Tensor processing unit (TPU) is an AI accelerator developed by Google for neural-network machine learning, using Google's own TensorFlow software

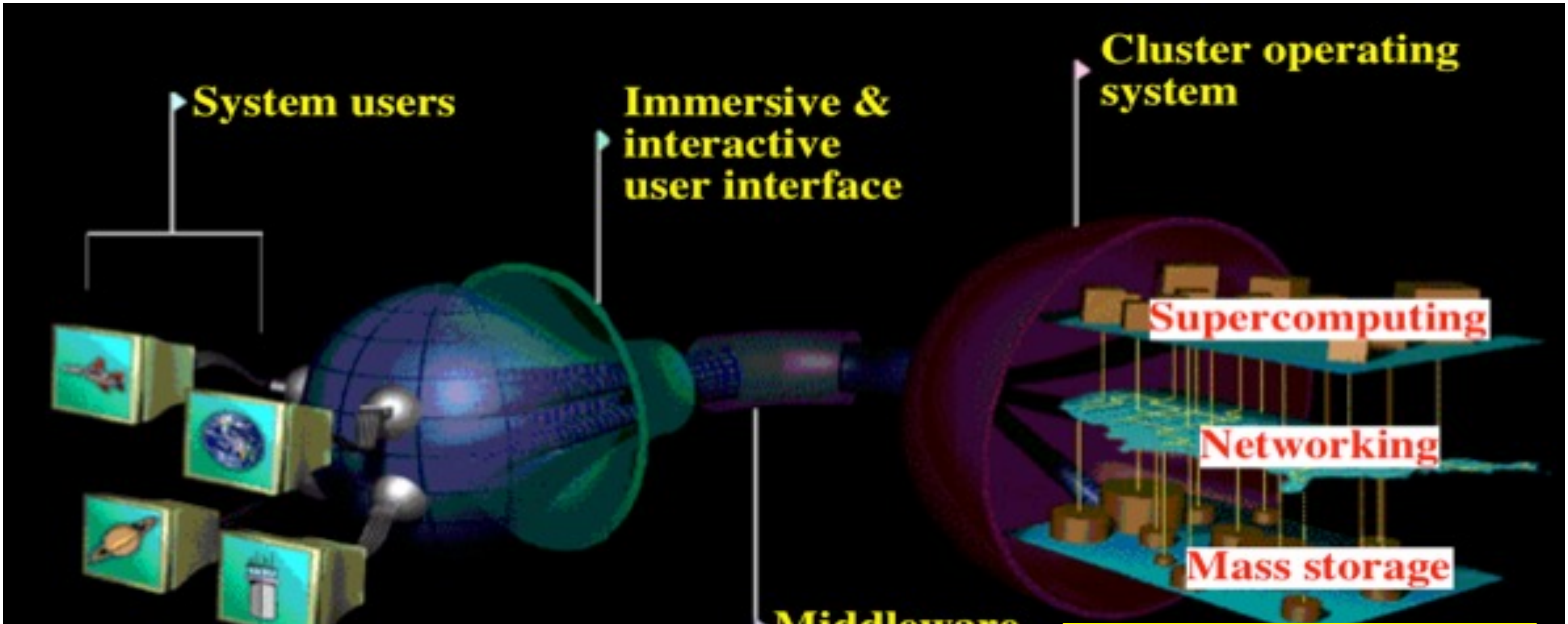
Google Cloud Says TPU-Powered Machine Learning Cluster Delivers 9 Exaflops Aggregate Power

May 12, 2022 by Doug Black

https://insidehpc.com

Grid Computing

- **World Wide Web:** Universal interface to digital library on the Internet
- **Information Grid:** Pervasive (from any place in the world at any time) access to everything (computing, mass storage, experimental equipments, distributed sensors, *etc*., on high-speed networks)

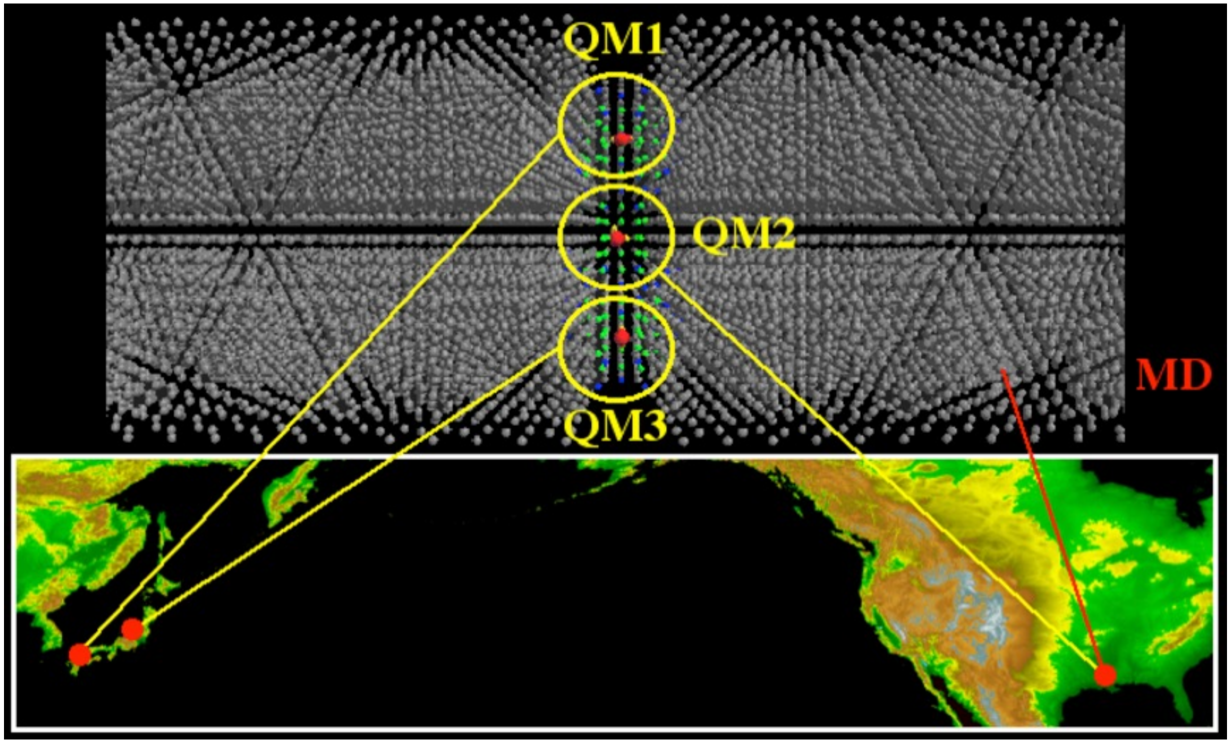


NASA NAS IPG

Global Collaborative Simulation

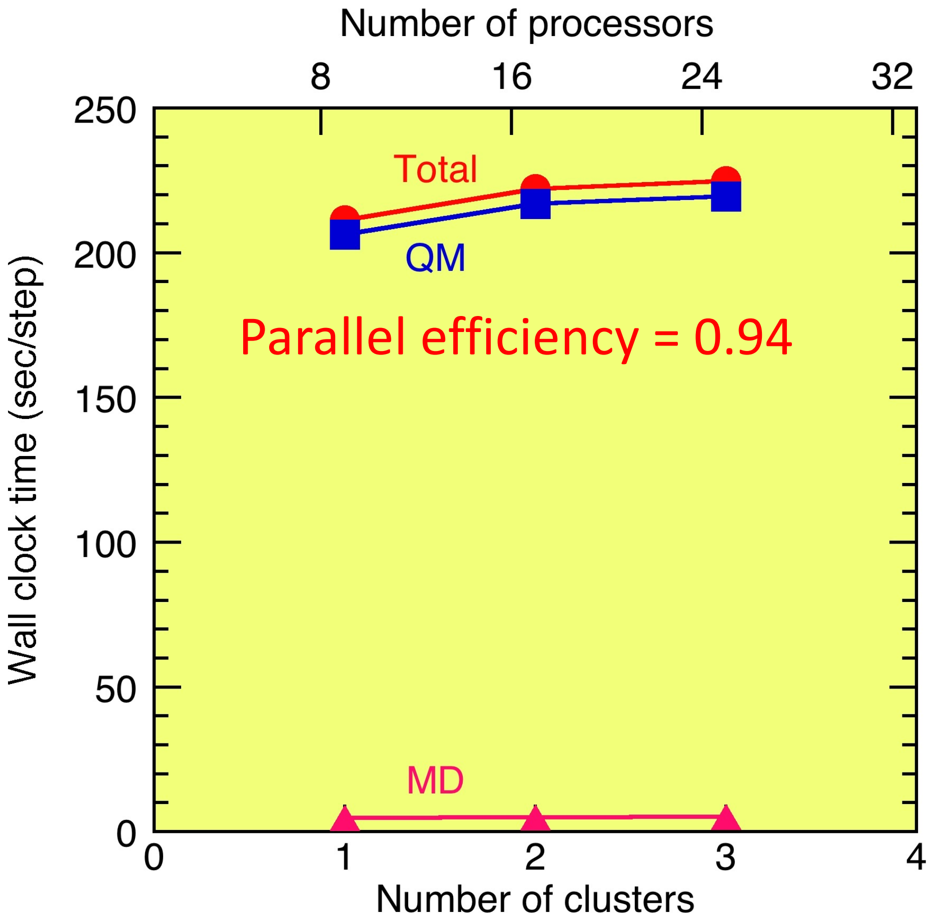
Multiscale MD/QMD simulation on a Grid of distributed PC clusters in the US & Japan

- Task decomposition (MPI Communicator) + spatial decomposition
- MPICH-G2/Globus



H. Kikuchi *et al*., “Collaborative simulation Grid: multiscale quantum-mechanical/classical atomistic simulations on distributed PC clusters in the US & Japan, *IEEE/ACM SC02*

Computation-Communication Overlap



$$\frac{\text{Earth's circumference}}{\text{Light speed}}$$

$$= \frac{40{,}000\ [\text{km}] = 4\times10^7 [\text{m}]}{3\times10^8 [\text{m/s}]} = 0.1\ \text{s} = 100\ \text{ms}$$

Try on Discovery:

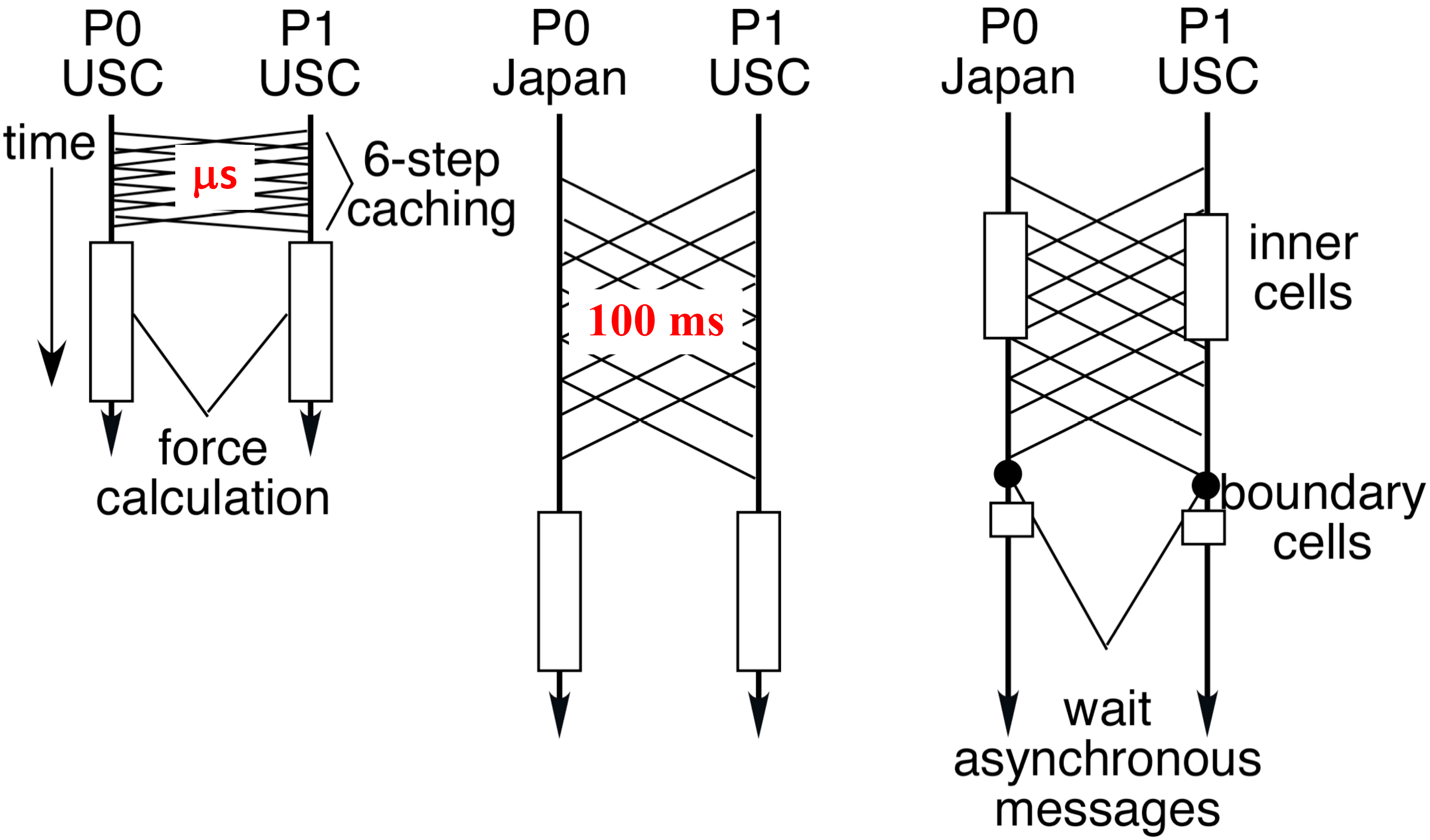
`traceroute www.u-tokyo.ac.jp`

vs. `ping hpc-transfer.usc.edu`

- **How to overcome 200 ms latency & 1 Mbps bandwidth?**
- **Computation-communication overlap: To hide the latency, the communications between the MD & QM processors have been overlapped with the computations using asynchronous messages**

Internode Optimization

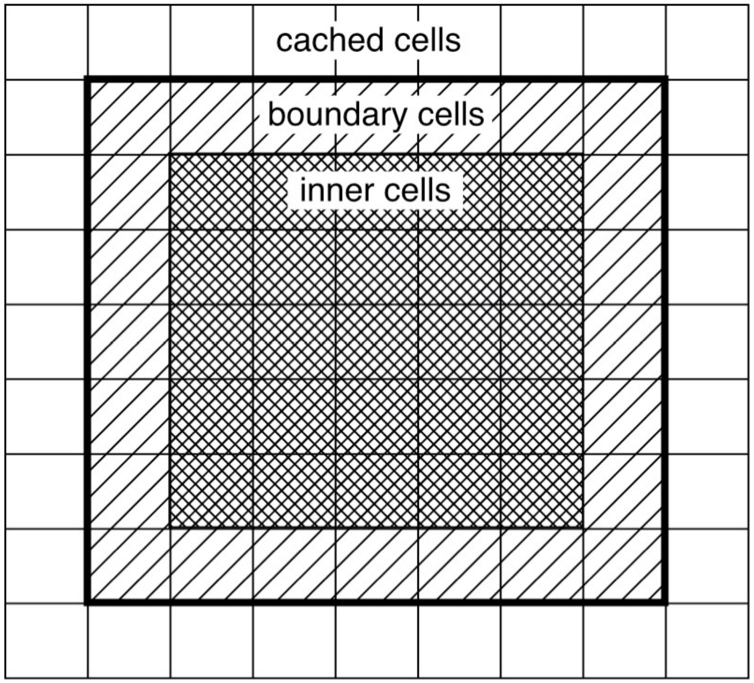
- **Communication bottleneck in metacomputing on a Grid**



Grid-Enabled MD Algorithm

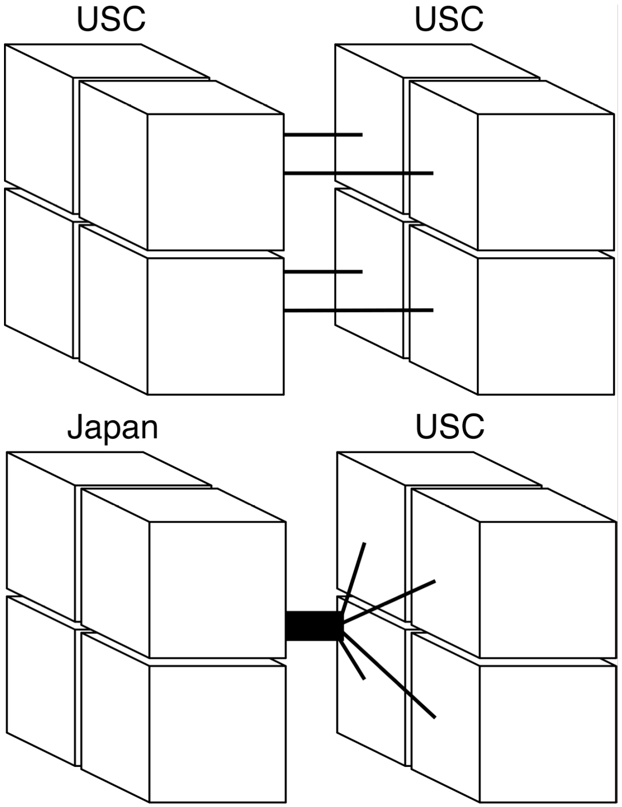
Grid MD algorithm:

1. asynchronous receive of cells to be cached **MPI_Irecv()**
2. send atomic coordinates in the boundary cells
3. compute forces for atoms in the inner cells
4. wait for the completion of the asynchronous receive **MPI_Wait()**
5. compute forces for atoms in the boundary cells



Renormalized Messages:

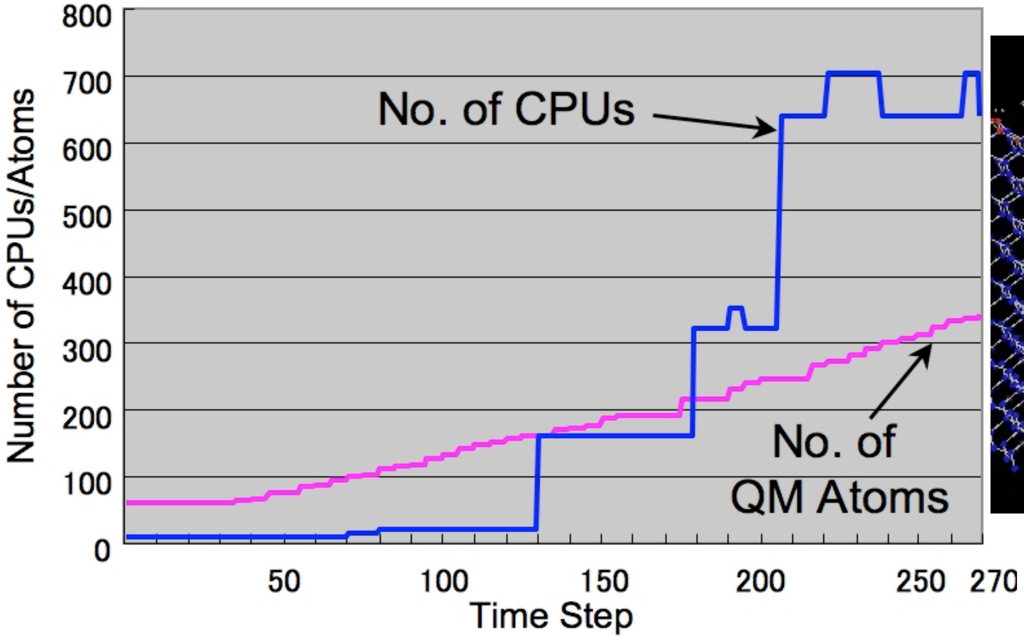
Latency can be reduced by composing a large cross-site message instead of sending all processor-to-processor messages

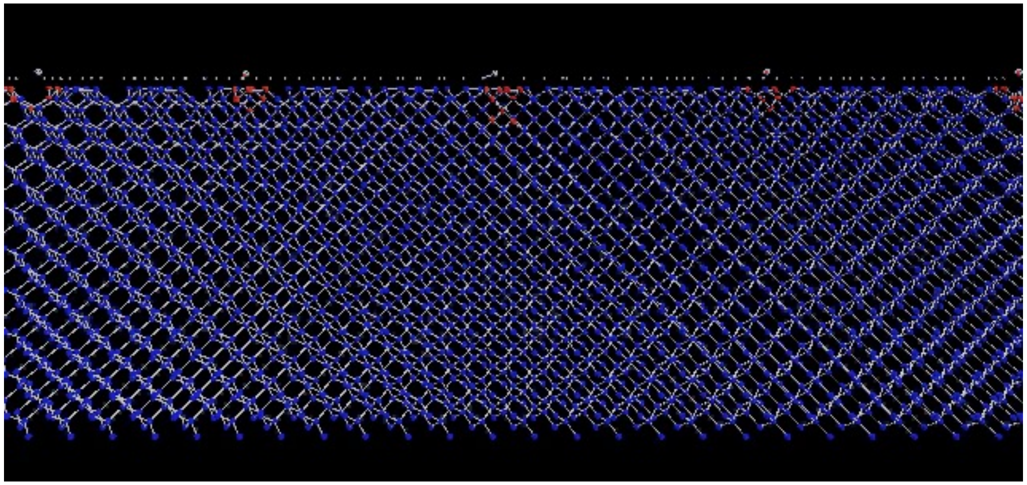


Sustainable Grid Supercomputing

- Sustained (> months) supercomputing (> 10^3 CPUs) on a Grid of geographically distributed supercomputers
- Hybrid Grid remote procedure call (GridRPC) + message passing (MPI) programming
- Dynamic allocation of computing resources on demand & automated migration due to reservation schedule & faults

Ninf-G GridRPC: ninf.apgrid.org; MPICH: www.mcs.anl.gov/mpi





Takemiya *et al.*, *IEEE/ACM SC06*
Song *et al.*, *IJCS* ('09)

Multiscale QM/MD simulation of high-energy beam oxidation of Si

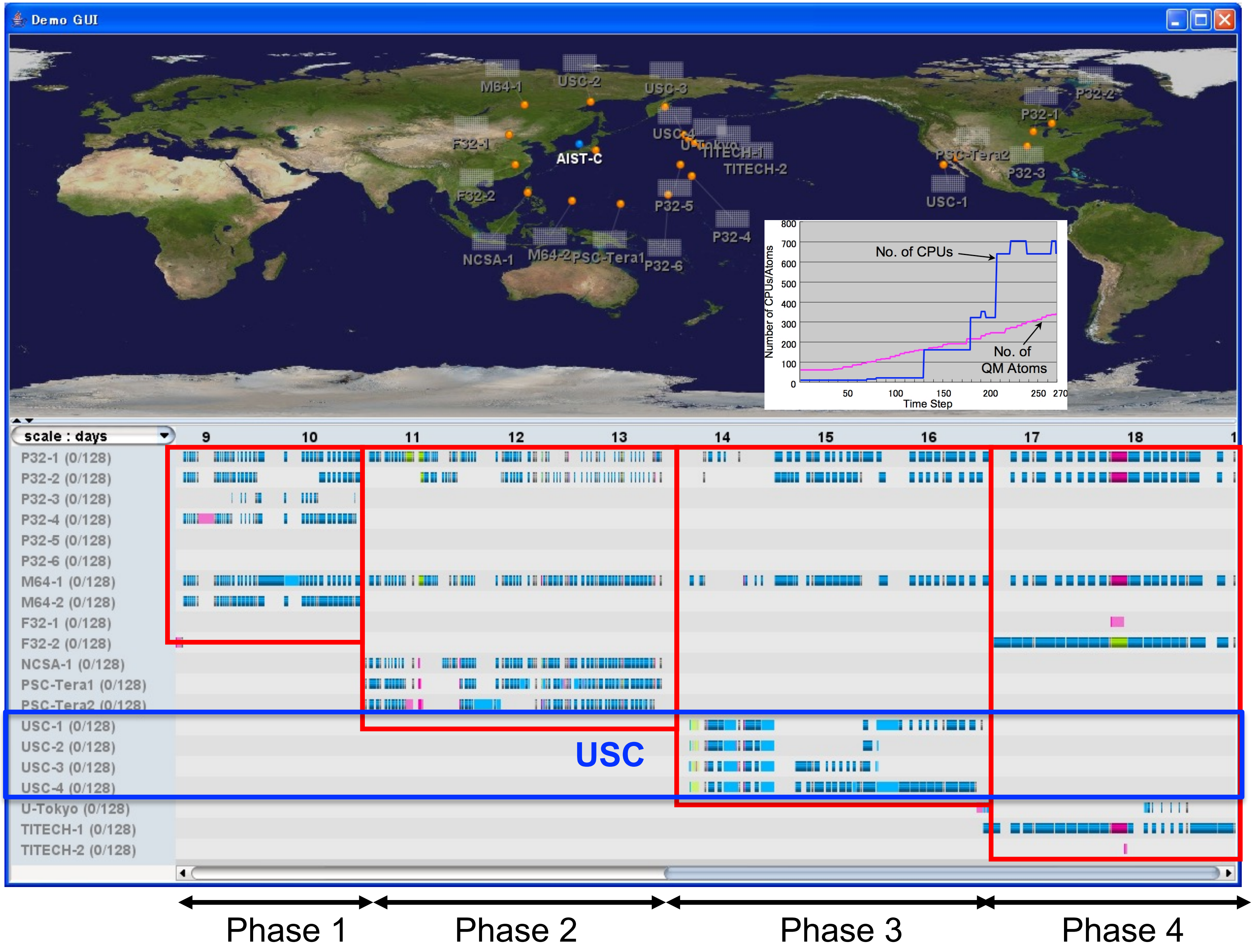
Grid Remote Procedure Call (RPC)

- **Simple RPC API (application program interface)**
- **Existing libraries & applications into Grid applications**
- **IDL (interface definition language) embodying call information, with minimal client-side management**

```
double A[n][n],B[n][n],C[n][n];      /* Data Declaration */
dmmul(n,A,B,C);                      /* Call local function */

grpc_function_handle_default(&hdl, “dmmul”);
grpc_call(hdl,n,A,B,C);   /* Call server-side routine */
```

- **Ninf–G Grid RPC system**
  http://ninf.apgrid.org

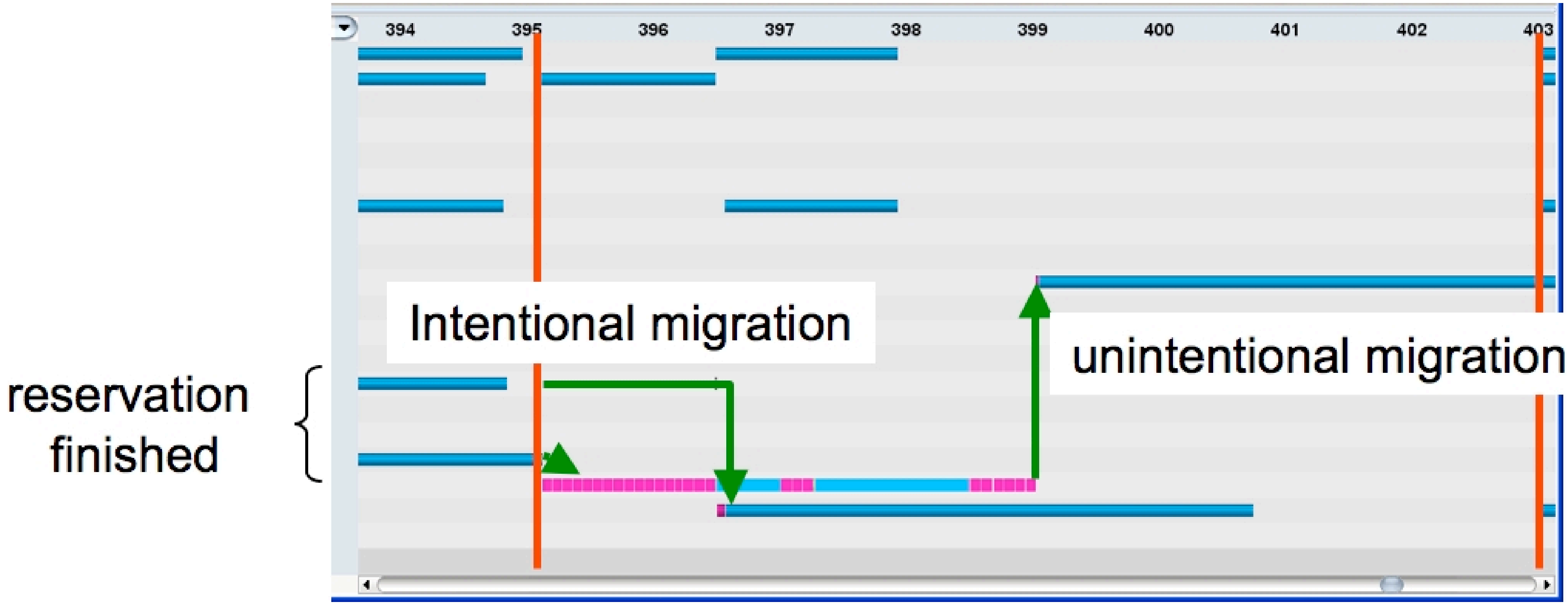
US-Japan Grid Testbed



Phase 1 Phase 2 Phase 3 Phase 4

Fault Tolerance

- **Automated migration in response to unexpected faults**



Current & Future Supercomputing

- **Won two DOE supercomputing awards to develop & deploy metascalable (“design once, scale on future platforms”) simulation algorithms**

786,432-core IBM Blue Gene/Q
281,088-core Intel Xeon Phi
560-node (2,240-GPU) AMD/NVIDIA
Polaris

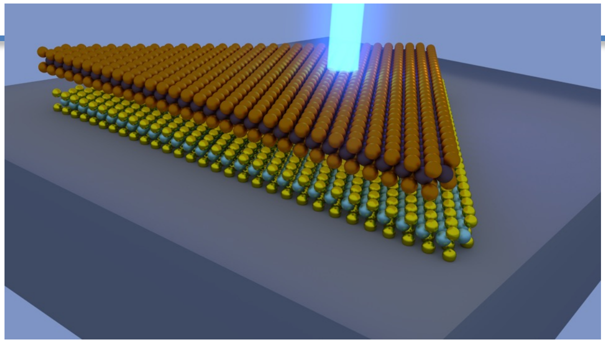
U.S. DEPARTMENT OF ENERGY
INCITE
LEADERSHIP COMPUTING

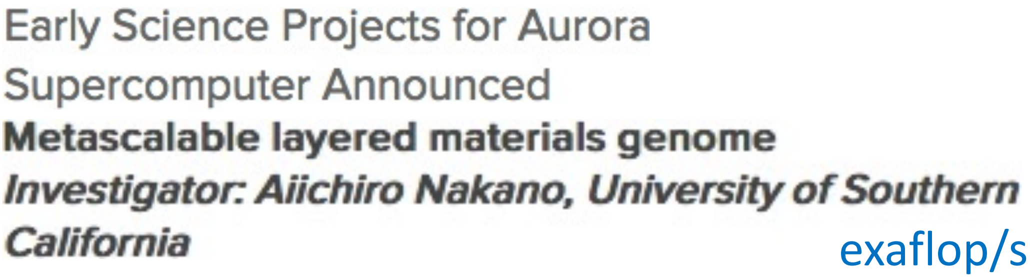
- **Atomistic simulations on million cores (pre-exascale)**

Innovative & Novel Computational Impact on Theory & Experiment

Title: AI-Guided Exascale Simulations of Quantum Materials Manufacturing and Control
PI and Co-PIs: Aiichiro Nakano–PI, Rajiv K. Kalia, Ken-ichi Nomura, Priya Vasishta

AURORA | EARLY SCIENCE PROGRAM





Early Science Projects for Aurora Supercomputer Announced

Metascalable layered materials genome

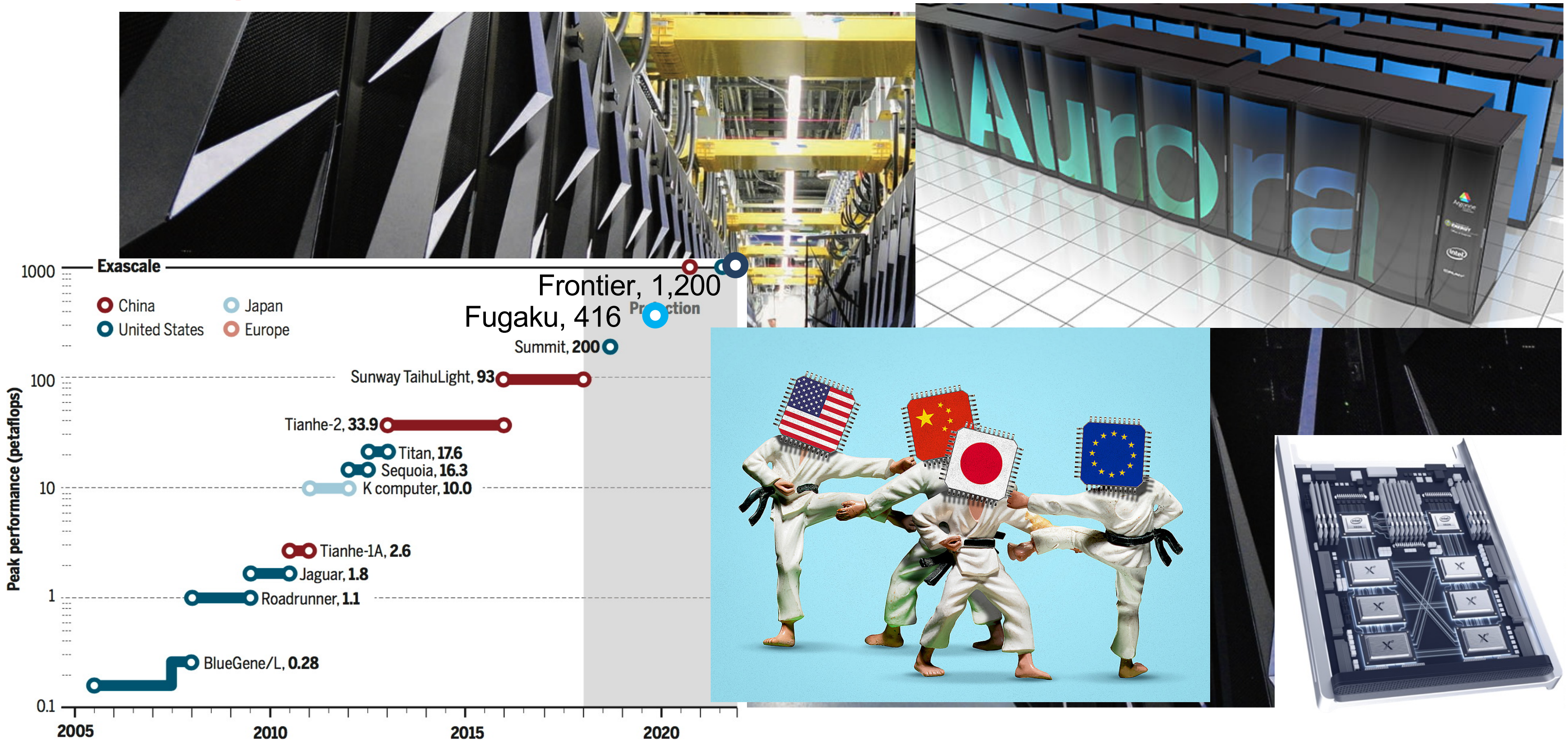
Investigator: Aiichiro Nakano, University of Southern California

1.01 exaflop/s
Intel Aurora

exaflop/s = 10^{18} mathematical operations per second

- **One of the initial simulation users of the next-generation DOE supercomputer**

CACS@Aurora in the Global Exascale Race



SUPERCOMPUTING

R. F. Service, *Science* 359, 617 ('18)

Design for U.S. exascale computer takes shape

Competition with China accelerates plans for next great leap in supercomputing power

Exa(peta)flop/s = 10^{18} (10^{15}) floating-point operations per second

By **Robert F. Service**

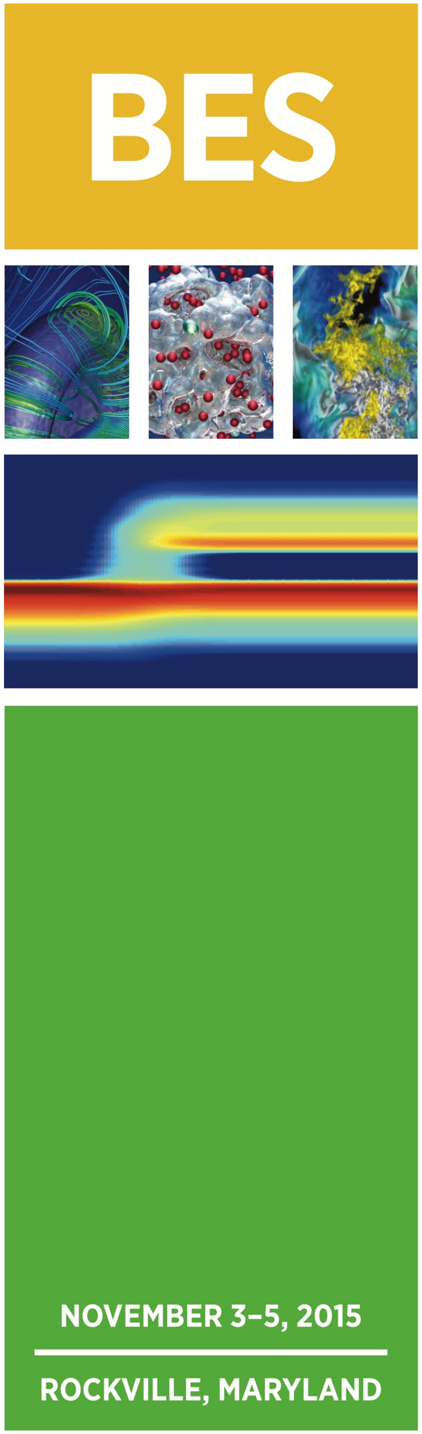
In 1957, the launch of the Sputnik satellite vaulted the Soviet Union to the lead in the space race and galvanized the United States. U.S. supercomputer researchers are today facing their own

Lemont, Illinois. That's 2 years earlier than planned. "It's a pretty exciting time," says Aiichiro Nakano, a physicist at the University of Southern California in Los Angeles who uses supercomputers to model materials made by layering stacks of atomic sheets like graphene.

pace reflects a change of strategy by DOE officials last fall. Initially, the agency set up a "two lanes" approach to overcoming the challenges of an exascale machine, in particular a potentially ravenous appetite for electricity that could require the output of a small nuclear plant.

https://www.tomshardware.com/news/two-chinese-exascale-supercomputers

Exa-leadership



BES

BASIC ENERGY SCIENCES

EXASCALE REQUIREMENTS REVIEW

An Office of Science review sponsored jointly by
Advanced Scientific Computing Research and Basic Energy Sciences

NOVEMBER 3–5, 2015

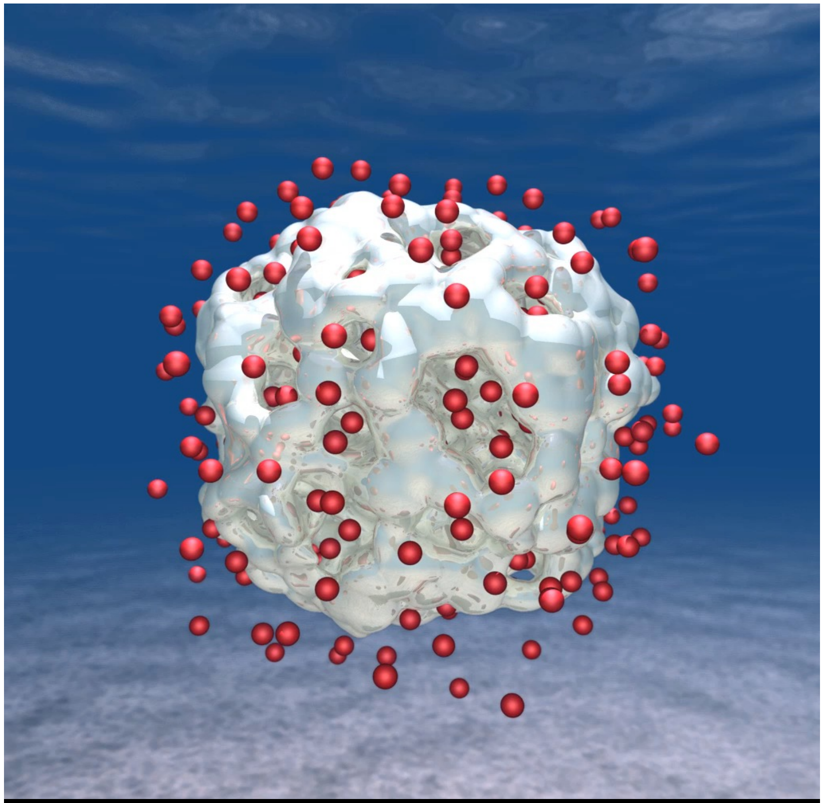
ROCKVILLE, MARYLAND

U.S. DEPARTMENT OF ENERGY

16,661-atom QMD

Shimamura *et al.*,
Nano Lett.
14, 4090 ('14)

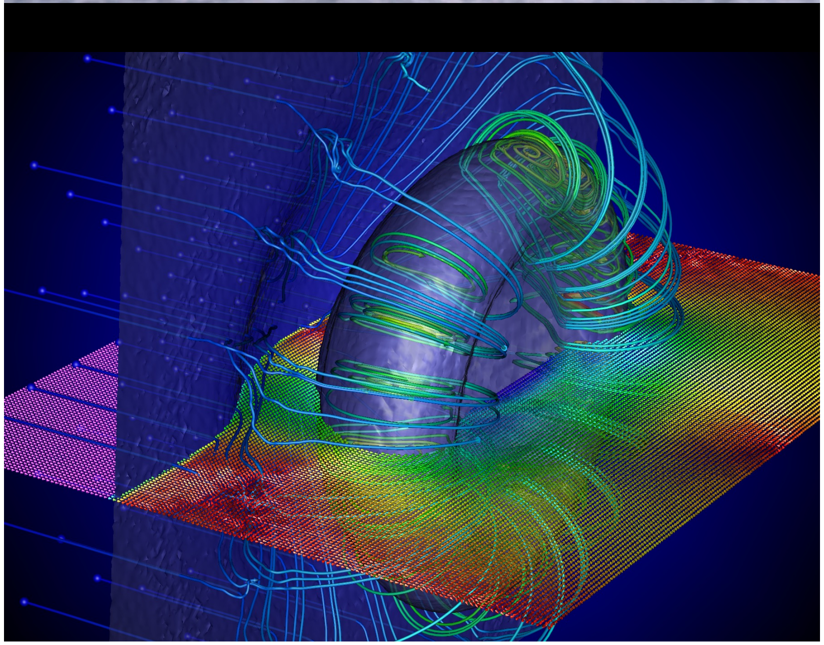
On-demand hydrogen production from water



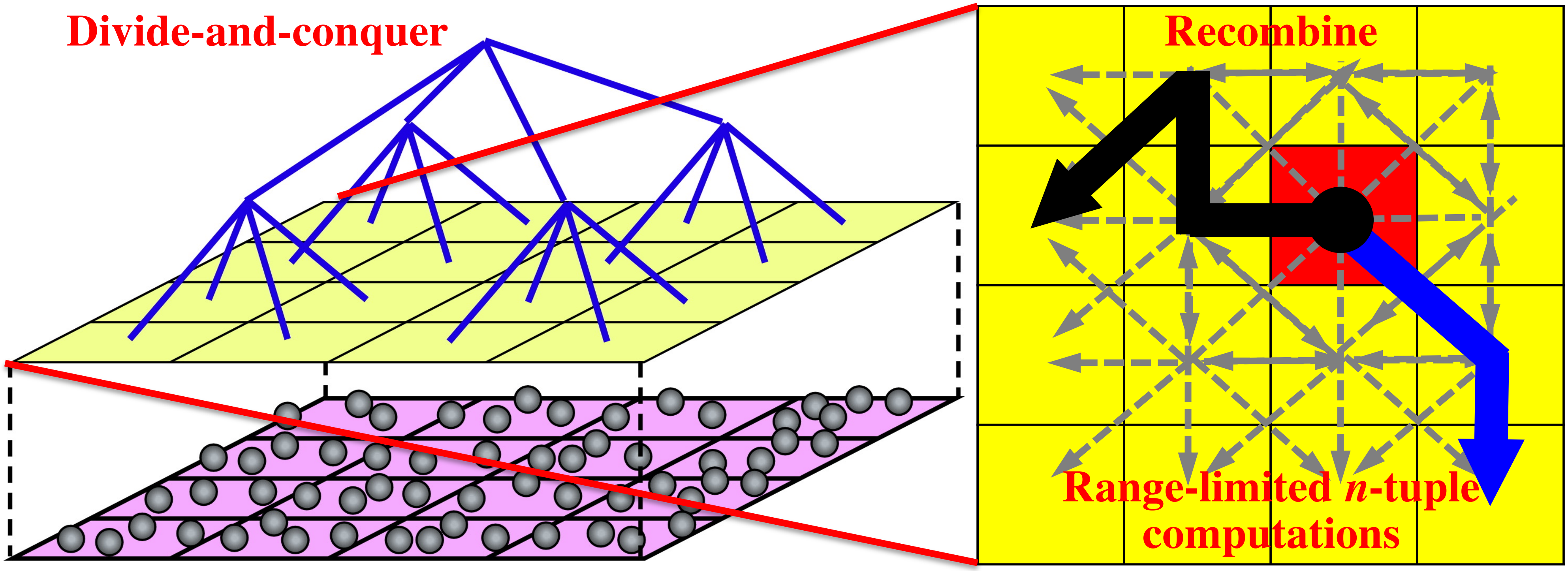
10^9-atom RMD

Shekhar *et al.*,
Phys. Rev. Lett.
111, 184503 ('13)

Fluid dynamics atom-by-atom



Divide-Conquer-Recombine (DCR) Engines



M. Kunaseth *et al*., *ACM/IEEE SC13*

See lecture on “shift-collapse” algorithm

- **Lean divide-&-conquer density functional theory (LDC-DFT) algorithm minimizes the prefactor of *O*(*N*) computational cost**

  F. Shimojo *et al*., *J. Chem. Phys.* **140**, 18A529 (’14); K. Nomura *et al*., *IEEE/ACM SC14*

- **Extended-Lagrangian reactive molecular dynamics (XRMD) algorithm eliminates the speed-limiting charge iteration**

  K. Nomura *et al*., *Comput. Phys. Commun*. **192**, 91 (’15)

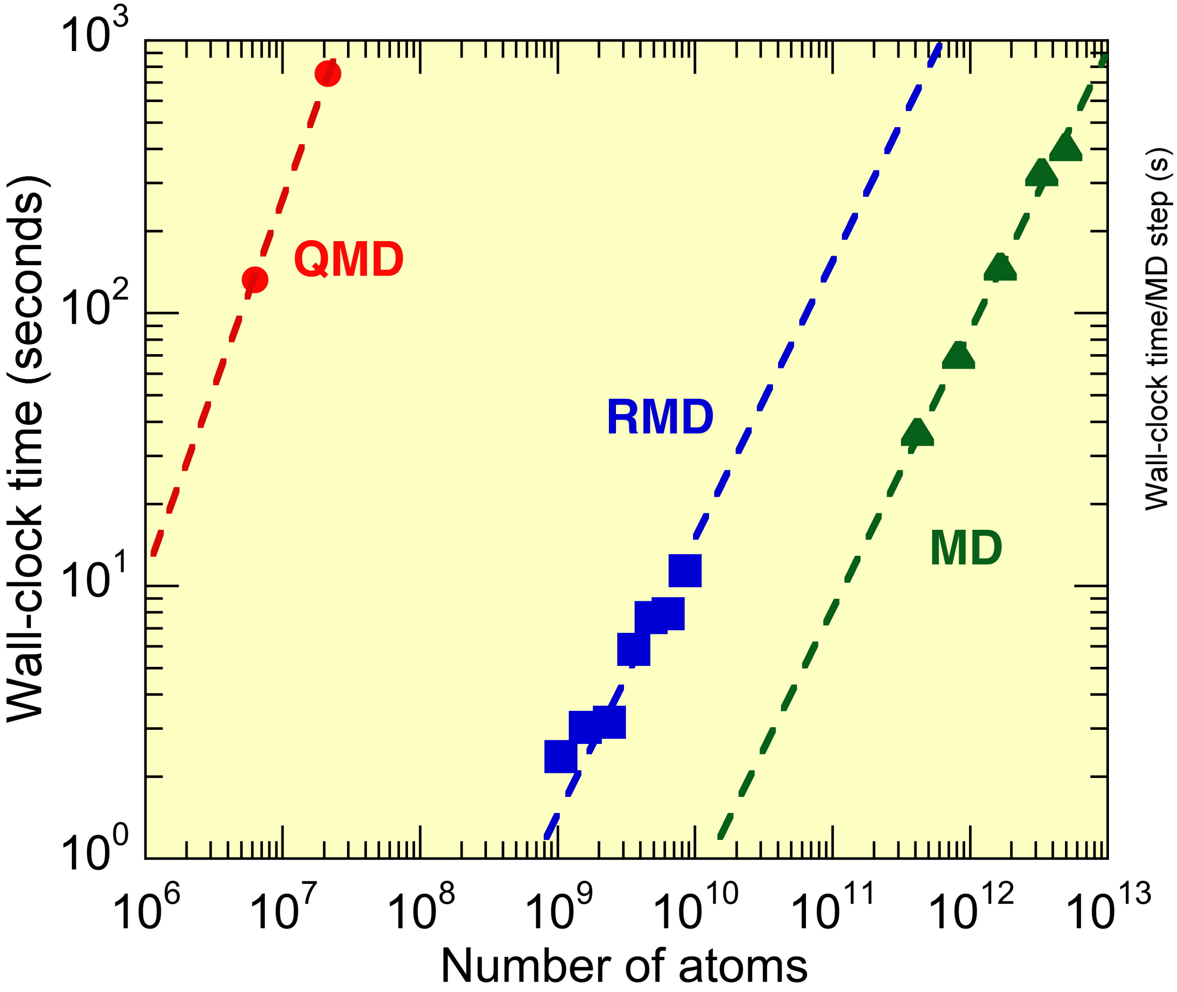
Divide-Conquer-(Re)combine

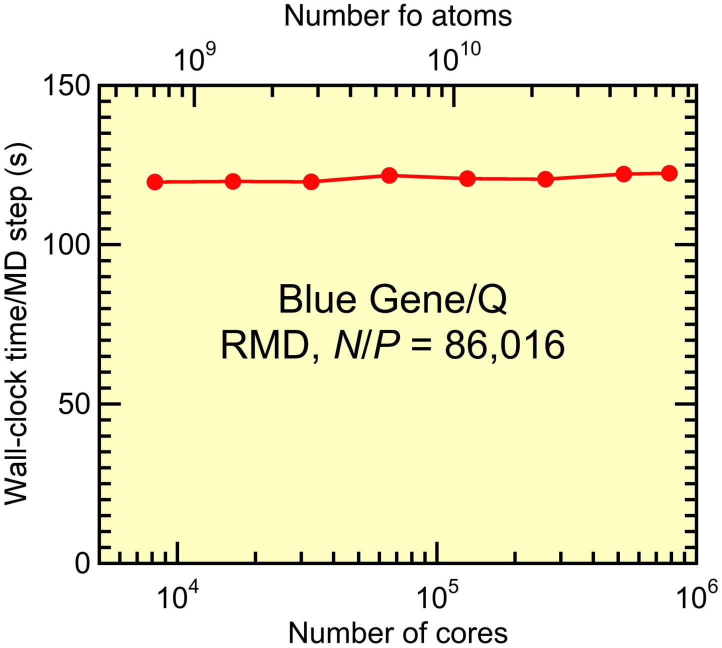
- "The first was to never accept anything as true which I could not accept as obviously true. The second was to divide each of the problems in as many parts as I should to solve them. The third, beginning with the simplest and easiest to understand matters, little by little, to the most complex knowledge. And the last resolution was to make my enumerations so complete and my reviews so general that I could be assured that I had not omitted anything." (René Descartes, *Discourse on Method*, 1637)
- 「モデルの分割ー再統合の方法の優れた点は、分割した要素的概念を、モデルの理解に役立つように再構成することができ、そこに創造の入り込む余地があるという点にある。」（福井謙一学問の創造、１９８７）

room for creativity

Kenichi Fukui [*Nobel Chemistry Prize*, '81]

Scalable Simulation Algorithm Suite





QMD (quantum molecular dynamics): DC-DFT

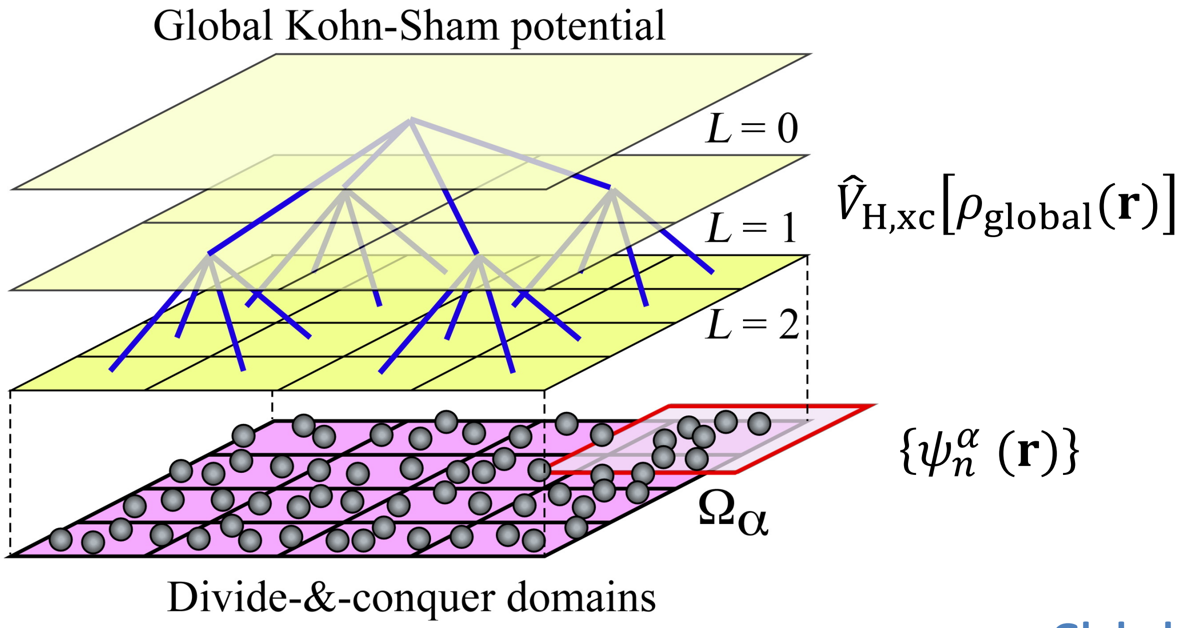
RMD (reactive molecular dynamics): F-ReaxFF

MD (molecular dynamics): MRMD

- 4.9 trillion-atom space-time multiresolution MD (MRMD) of SiO_2
- 67.6 billion-atom fast reactive force-field (F-ReaxFF) RMD of RDX
- 39.8 trillion grid points (50.3 million-atom) DC-DFT QMD of SiC

parallel efficiency 0.984 on 786,432 Blue Gene/Q cores

Divide-&-Conquer Density Functional Theory


Global Kohn-Sham potential
$L = 0$
$\hat{V}_{\text{H,xc}}[\rho_{\text{global}}(\mathbf{r})]$
$L = 1$
$L = 2$
$\{\psi_n^\alpha(\mathbf{r})\}$
Ω_α
Divide-&-conquer domains

- **Overlapping spatial domains:** $\Omega = \cup_\alpha \Omega_\alpha$
- **Domain Kohn-Sham equations**

$$\left(-\frac{1}{2}\nabla^2 + \hat{V}_{\text{ion}} + \hat{V}_{\text{H,xc}}\left[\boxed{\rho_{\text{global}}(\mathbf{r})}\right]\right)\psi_n^\alpha(\mathbf{r}) = \epsilon_n^\alpha \boxed{\psi_n^\alpha(\mathbf{r})}$$

Global-local self-consistent field (SCF) iteration

- **Global & domain electron densities**

$$\boxed{\rho_{\text{global}}(\mathbf{r})} = \Sigma_\alpha\, p_\alpha(\mathbf{r})\boxed{\rho_\alpha(\mathbf{r})} \longleftarrow \boxed{\rho_\alpha(\mathbf{r})} = \Sigma_n \boxed{|\psi_n^\alpha|}^2\, \Theta(\mu - \epsilon_n^\alpha)$$

Domain support function

$$\Sigma_\alpha\, p_\alpha(\mathbf{r}) = 1$$

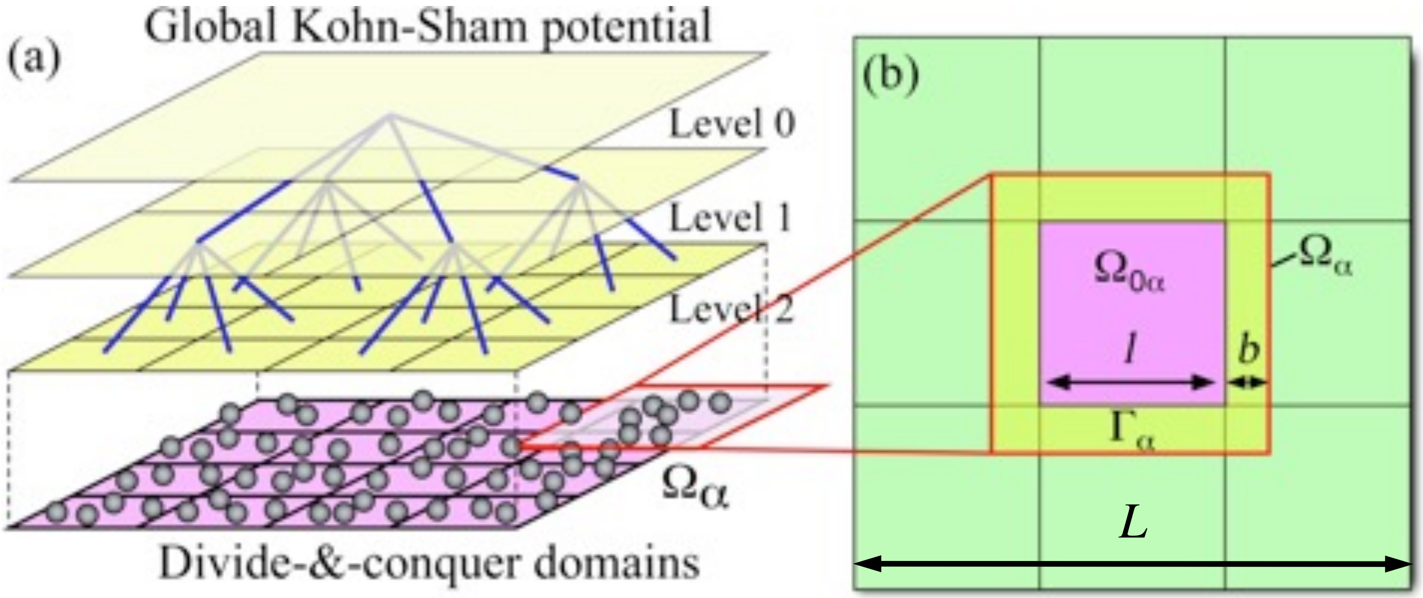
Global chemical potential

$$N = \int d\mathbf{r}\, \rho_{\text{global}}(\mathbf{r})$$

cf. subsystem DFT [W. Mi *et al.*, *Comput. Phys. Commun.* **269**, 108122 ('21)]

Optimization of Divide-&-Conquer DFT

- **Computational parameters of DC-DFT = domain size (*l*) + buffer thickness (*b*)**



- **Complexity analysis to optimize the domain size *l***

$$l_* = \operatorname{argmin}\left(T_{\mathrm{comp}}(l)\right) = \operatorname{argmin}\left(\left(\frac{L}{l}\right)^3 (l+2b)^{3\nu}\right) = \frac{2b}{\nu - 1}$$

Per-domain computational complexity of DFT = $O(n^\nu)$: ν = 2 or 3 (n < or > 10^3)

- **Error analysis: Buffer thickness *b* is dictated by the accuracy requirement**

$$b = \lambda \ln\left(\max\{|\Delta\rho_\alpha(\mathbf{r})| \,|\, \mathbf{r} \in \partial\Omega_\alpha\}\right)/\varepsilon\langle\rho_\alpha(\mathbf{r})\rangle \qquad |\Delta\rho| e^{-b/\lambda} = \varepsilon\langle\rho\rangle$$

Decay length $\quad \rho_\alpha(\mathbf{r}) - \rho_{\mathrm{global}}(\mathbf{r}) \quad$ Error tolerance

cf. quantum nearsightedness [Kohn, *Phys. Rev. Lett*. **76**, 3168 ('96); Prodan & Kohn, *P. Nat. Acad. Sci*. **102**, 11635 ('05)]

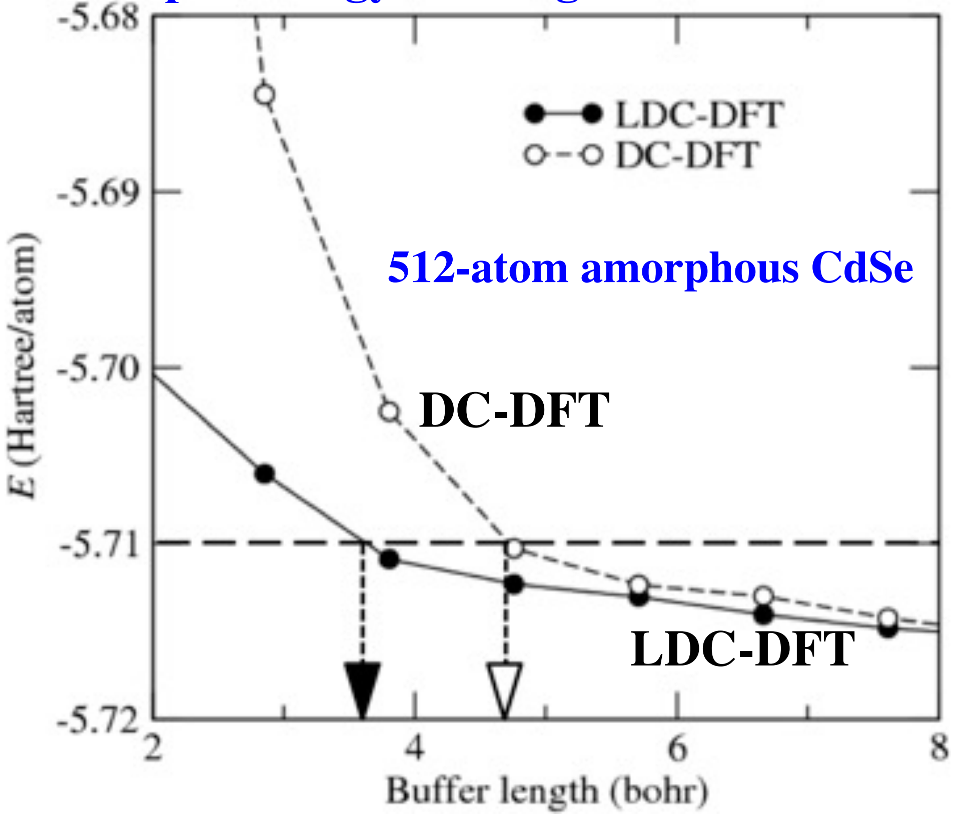
Lean Divide-&-Conquer (LDC) DFT

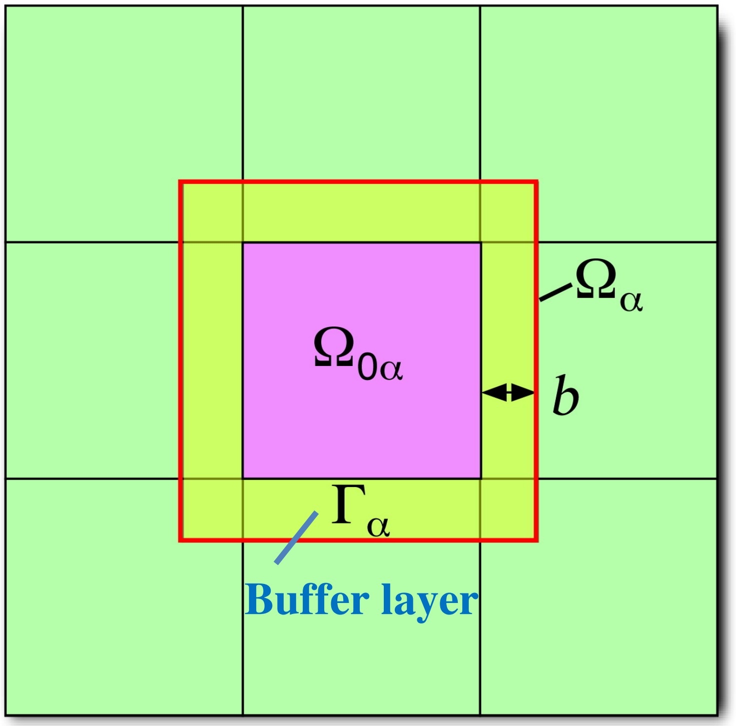
- **Density-adaptive boundary potential to reduce the *O*(*N*) prefactor**

local approximation

$$v_\alpha^{\mathrm{bc}}(\mathbf{r}) = \int d\mathbf{r}' \frac{\partial v(\mathbf{r})}{\partial \rho(\mathbf{r}')} \left(\rho_\alpha(\mathbf{r}') - \rho_{\mathrm{global}}(\mathbf{r}')\right) \cong \frac{\rho_\alpha(\mathbf{r}) - \rho_{\mathrm{global}}(\mathbf{r})}{\xi}$$

- **More rapid energy convergence of LDC-DFT compared with nonadaptive DC-DFT**



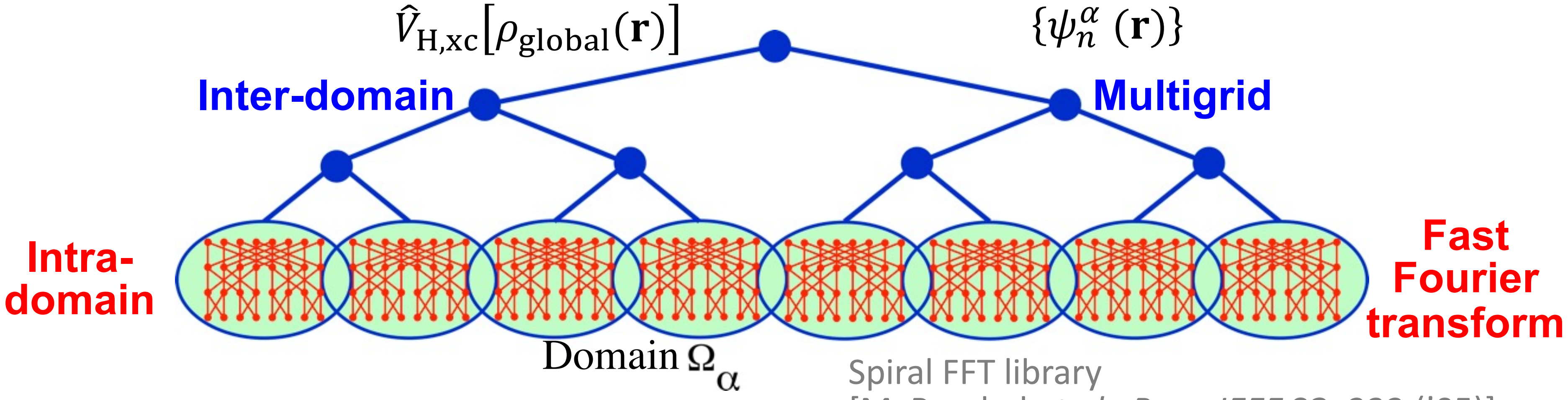


- **Factor 2.03 (for ν = 2) ~ 2.89 (for ν = 3) reduction of the computational cost with an error tolerance of 5×10^{-3} a.u. (per-domain complexity: n^ν)**

F. Shimojo *et al.*, *J. Chem. Phys.* **140**, 18A529 ('14);
Phys. Rev. B **77**, 085103 ('08); *Comput. Phys. Commun*. **167**, 151 ('05)

Hierarchical Computing

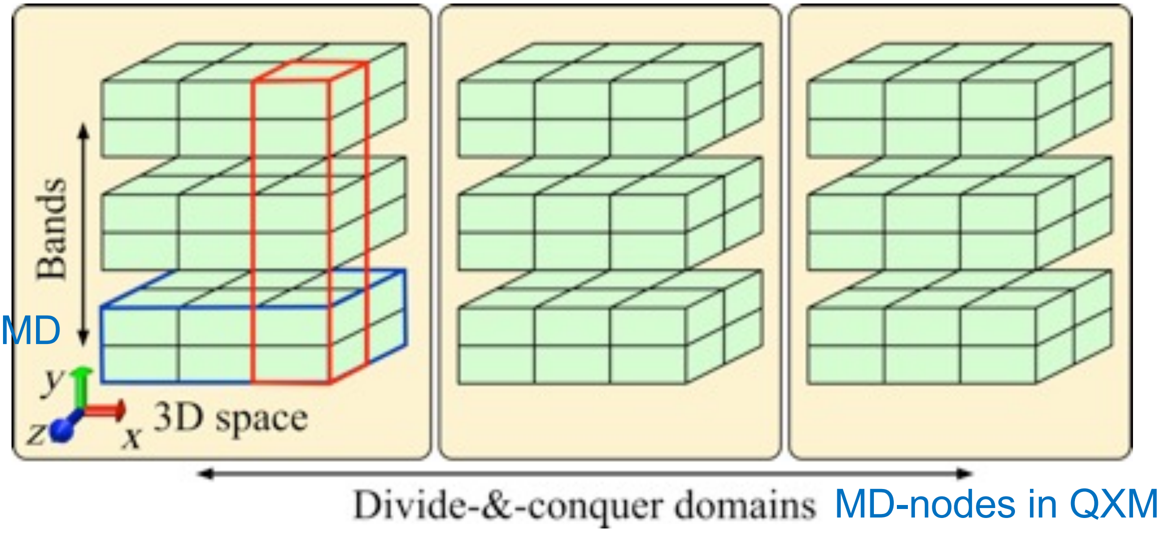
- **Globally scalable (real-space multigrid) + locally fast (plane wave) electronic solver**


$\hat{V}_{\mathrm{H,xc}}[\rho_{\mathrm{global}}(\mathbf{r})]$
$\{\psi_n^{\alpha}(\mathbf{r})\}$
Inter-domain
Multigrid
Intra-domain
Fast Fourier transform
Domain Ω_α

Spiral FFT library
[M. Puschel *et al.*, *Proc. IEEE* **93**, 232 ('05)]

cf. globally- sparse-yet-locally-dense eigensolver [J. H. Lam *et al., Nature Commun*. **15**, 3479 ('24)]

- **Hierarchical band (*i.e.*, Kohn-Sham orbital) + space + domain (BSD) decomposition**



QM-nodes in QXMD

MD-nodes in QXMD software

Key Skill: Scalability Analysis

- **Parallel computing = Solving a big problem (*W*) in a short time (*T*) using many processors (*P*)**

- **How *W*, *T* & *P* scale with each other?**
- **How to define the efficiency of a parallel program?**

See Grama *et al*.,
Chap. 5—Analytical modeling of parallel programs

Parallel Efficiency

- **Execution time: *T*(*W*,*P*)**
  ***W*: Workload**
  ***P*: Number of processors**

- **Speed:** $$S(W,P) = \frac{W}{T(W,P)}$$

- **Speedup:** $$S_P = \frac{S(W_P,P)}{S(W_1,1)} = \frac{W_P T(W_1,1)}{W_1 T(W_P,P)}$$

- **Efficiency:** $$E_P = \frac{S_P}{P} = \frac{W_P T(W_1,1)}{P W_1 T(W_P,P)}$$

Ideal speedup

How to scale W_P with *P*?

See https://aiichironakano.github.io/cs596.html

Fixed Problem-Size (Strong) Scaling

$W_P = W$—constant (strong scaling)

- **Speedup:** $S_P = \dfrac{T(W,1)}{T(W,P)}$

$$S_P = \frac{S(W_P,P)}{S(W_1,1)} = \frac{W_P T(W_1,1)}{W_1 T(W_P,P)}$$

- **Efficiency:** $E_P = \dfrac{T(W,1)}{PT(W,P)}$

$$E_P = \frac{S_P}{P} = \frac{W_P T(W_1,1)}{PW_1 T(W_P,P)}$$

Solving the same problem faster using more processors!

- **Amdahl's law:** ***f*** **(= sequential fraction of the workload) limits the asymptotic speedup**

$$S_P = \frac{T(W,1)}{T(W,P)} \leq P$$

$$T(W,P) = fT(W,1) + \frac{(1-f)T(W,1)}{P}$$

$$\therefore S_P = \frac{T(W,1)}{T(W,P)} = \frac{1}{f + (1-f)/P}$$

$$\therefore S_P \to \frac{1}{f} \quad (P \to \infty)$$

Isogranular (Weak) Scaling

$W_P = Pw$ **(weak scaling)**
w **= constant workload per processor (granularity)**

- **Speedup:** $S_P = \dfrac{S(P \bullet w, P)}{S(w,1)} = \dfrac{P \bullet w / T(P \bullet w, P)}{w / T(w,1)} = \dfrac{P \bullet T(w,1)}{T(P \bullet w, P)}$
- **Efficiency:** $E_P = \dfrac{S_P}{P} = \dfrac{T(w,1)}{T(P \bullet w, P)}$

$$S_P = \frac{S(W_P, P)}{S(W_1, 1)} = \frac{W_P T(W_1, 1)}{W_1 T(W_P, P)}$$

$$E_P = \frac{S_P}{P} = \frac{W_P T(W_1, 1)}{P W_1 T(W_P, P)}$$

Solving larger problems within the same time using more processors!

$$E_P = \frac{T(w,1)}{T(Pw,P)} \leq 1$$

Analysis of Parallel MD

- **Parallel execution time:**
  Workload ∝ Number of atoms, *N* (linked-list cell algorithm)

MPI_Allreduce()

$$T(N,P) = T_{\mathrm{comp}}(N,P) + T_{\mathrm{comm}}(N,P) + T_{\mathrm{global}}(P)$$

$$= a\frac{N}{P} + b\left(\frac{N}{P}\right)^{2/3} + c\log P$$

$$\overbrace{6}^{\text{facets}}\ \overbrace{\frac{L^2}{P^{2/3}}r_{\mathrm{c}}}^{\text{cached volume}} \times \overbrace{\rho}^{\text{atom density}}$$

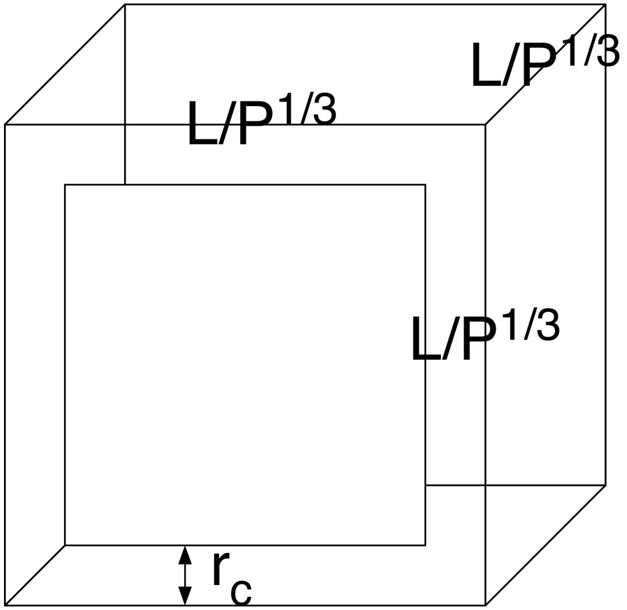
$$= 6r_{\mathrm{c}}\frac{N^{2/3}/\rho^{2/3}}{P^{2/3}}\rho$$

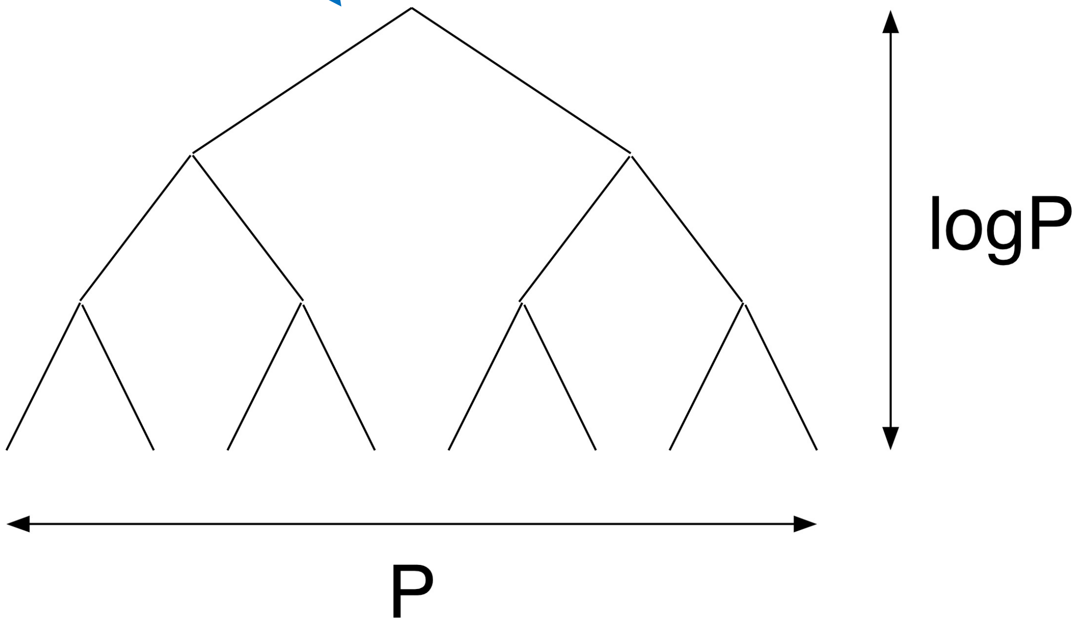
$$= 6r_{\mathrm{c}}\rho^{1/3}\left(\frac{N}{P}\right)^{2/3}$$

$$\left(\because \frac{N}{L^3} = \rho \Rightarrow L^2 = \frac{N^{2/3}}{\rho^{2/3}}\right)$$

Surface-to-volume ratio

Divide & conquer tree-height





Fixed Problem-Size (Strong) Scaling

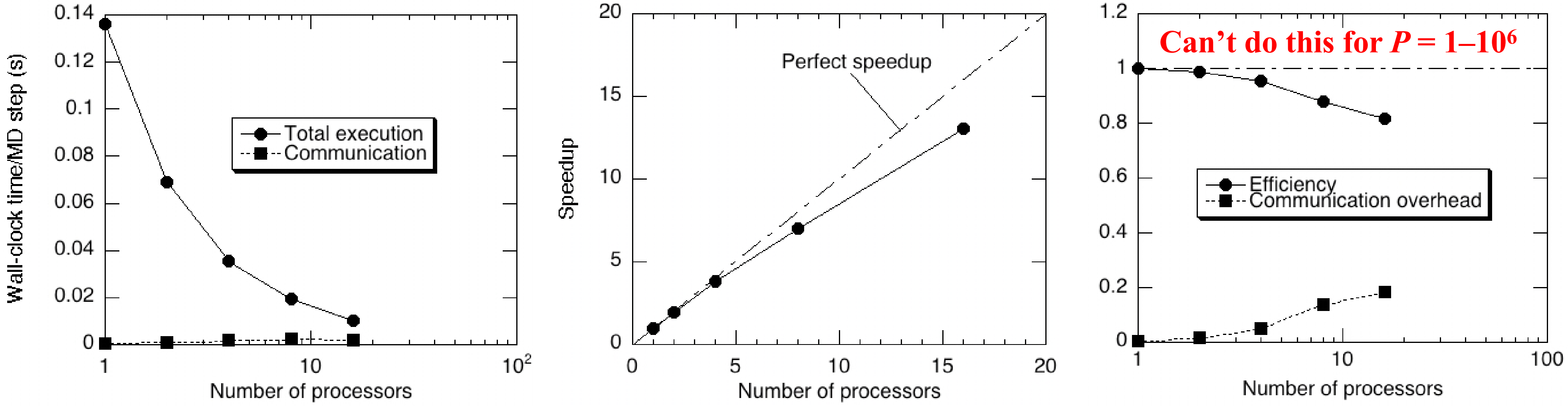
- **Speedup:**

$$S_P = \frac{T(N,1)}{T(N,P)} = \frac{aN}{aN/P + b(N/P)^{2/3} + c\log P}$$

$$= \frac{P}{1 + \frac{b}{a}\left(\frac{P}{N}\right)^{1/3} + \frac{c}{a}\frac{P\log P}{N}}$$

- **Efficiency:**

$$E_P = \frac{S_P}{P} = \frac{1}{1 + \frac{b}{a}\boxed{\left(\frac{P}{N}\right)^{1/3}} + \frac{c}{a}\frac{\boxed{P\log P}}{N}}$$



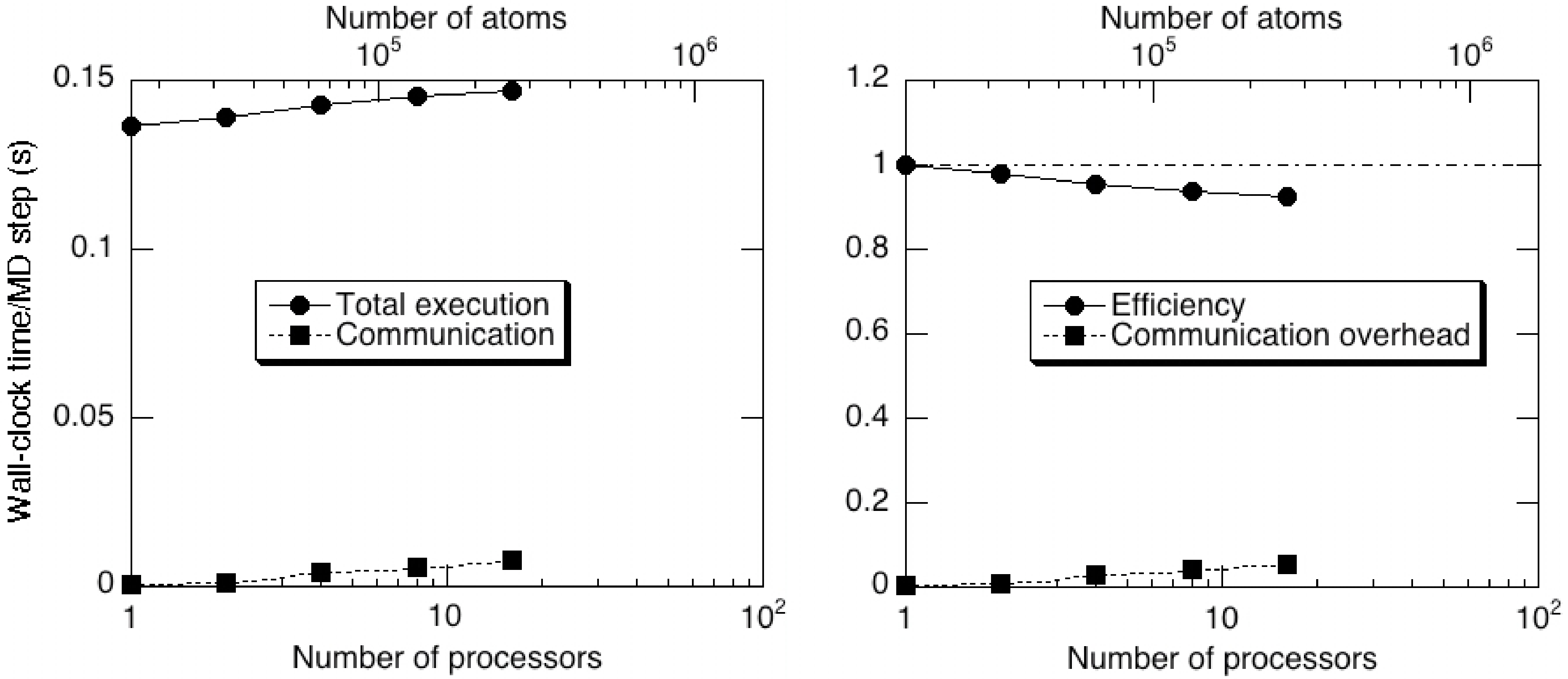
Can't do this for P = 1–10^6

pmd.c: N = 16,384, on HPC (predecessor of CARC)

Isogranular (Weak) Scaling of Parallel MD

- ***n* = *N*/*P* = constant: doable for arbitrarily large *P***
- **Efficiency:**

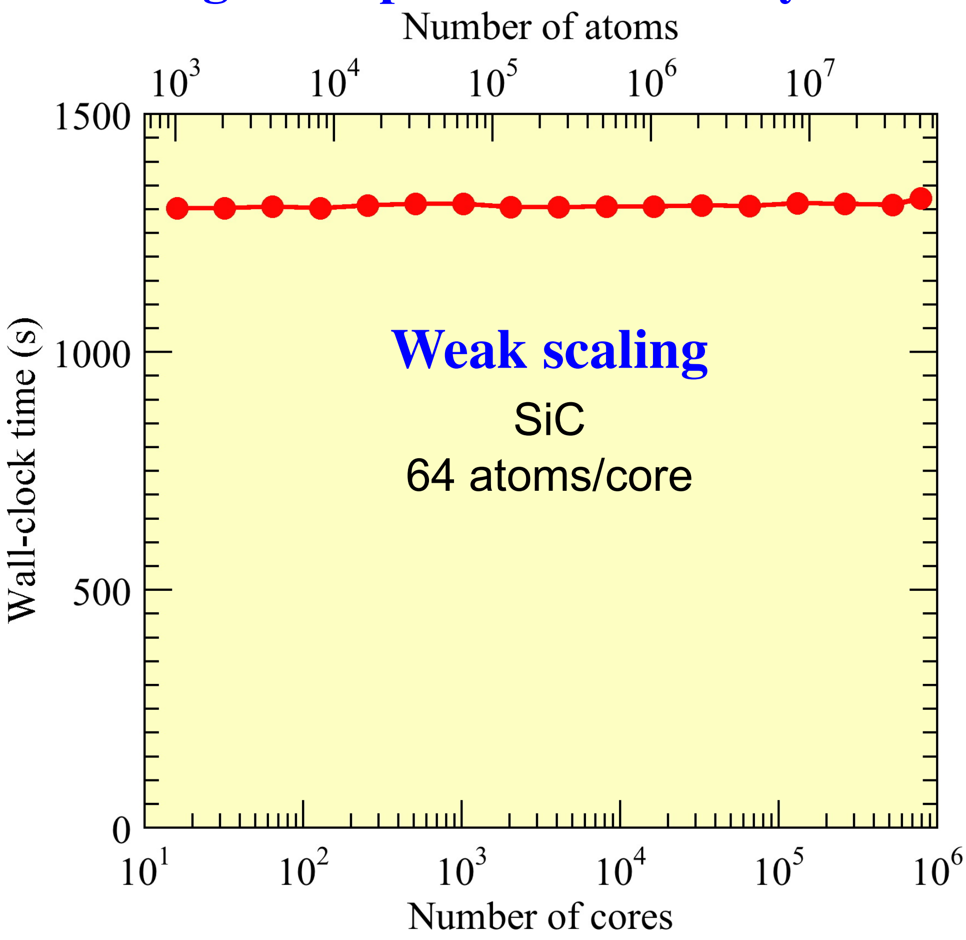
$$E_P = \frac{T(n,1)}{T(nP,P)} = \frac{an}{an + bn^{2/3} + c\log P} = \frac{1}{1 + \frac{b}{a}n^{-1/3} + \frac{c}{an}\boxed{\log P}}$$

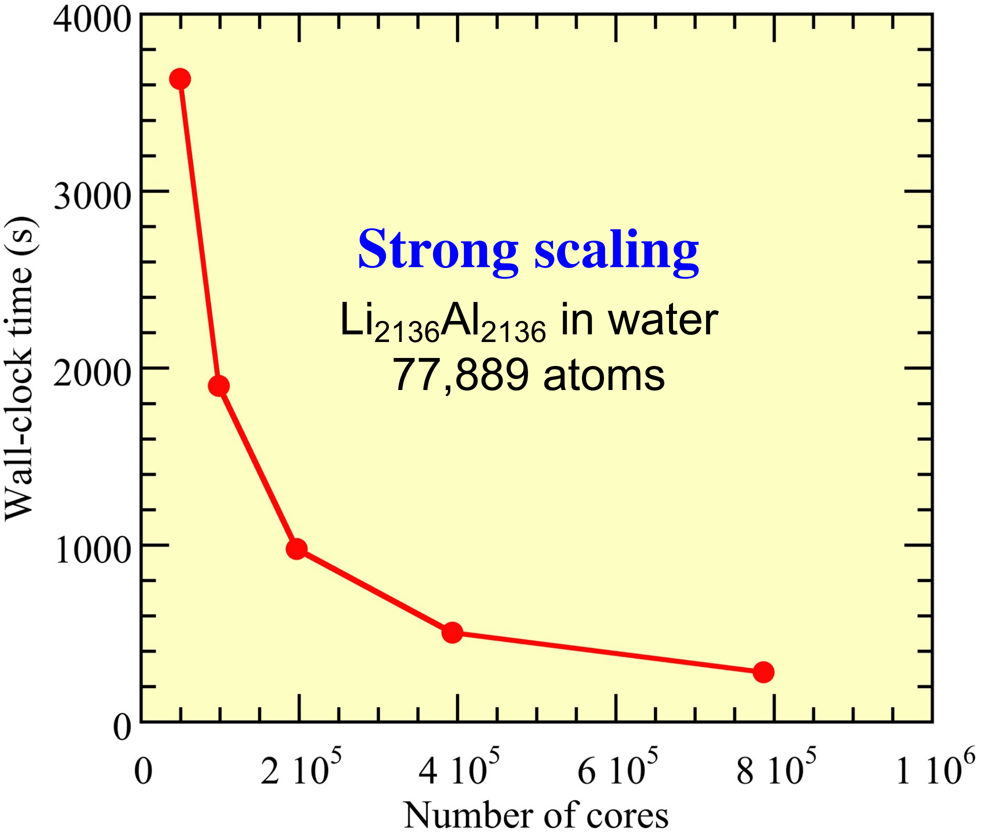


pmd.c: *N*/*P* = 16,384, on HPC (predecessor of CARC)

Parallel Performance of QXMD

- **Weak-scaling parallel efficiency is 0.984 on 786,432 Blue Gene/Q cores for a 50,331,648-atom SiC system**
- **Strong-scale parallel efficiency is 0.803 on 786,432 Blue Gene/Q cores**





- **62-fold reduction of time-to-solution** [441 s/SCF-step for 50.3M atoms] **from the previous state-of-the-art** [55 s/SCF-step for 102K atoms, Osei-Kuffuor *et al.*, *PRL* ’14]

K. Nomura *et al.*, *IEEE/ACM Supercomputing, SC14* (’14)

BLASification

- **Transform from band-by-band to all-band computations to utilize a matrix-matrix subroutine (DGEMM) in the level 3 basic linear algebra subprograms (BLAS3) library**
- **Algebraic transformation of computations**

Example: Nonlocal pseudopotential operation

D. Vanderbilt, *Phys. Rev. B* **41**, 7892 ('90)

$$\hat{v}_{\mathrm{nl}}|\psi_n^\alpha\rangle = \sum_{I}^{N_{\mathrm{atom}}} \sum_{ij}^{L_{\max}} |\beta_{i,I}\rangle D_{ij,I}\langle\beta_{j,I}|\psi_n^\alpha\rangle \quad (n = 1, \ldots, N_{\mathrm{band}})$$

$$\boldsymbol{\Psi} = \left[|\psi_1^\alpha\rangle, \ldots, |\psi_{N_{\mathrm{band}}}^\alpha\rangle\right] \ \tilde{\mathbf{B}}(i) = \left[|\beta_{i,1}\rangle, \ldots, |\beta_{i,N_{\mathrm{atom}}}\rangle\right] \ \left[\tilde{\mathbf{D}}(i,j)\right]_{I,J} = D_{ij,I}\delta_{IJ}$$

$$\hat{v}_{\mathrm{nl}}\boldsymbol{\Psi} = \sum_{i,j}^{L} \tilde{\mathbf{B}}(i)\tilde{\mathbf{D}}(i,j)\tilde{\mathbf{B}}(j)^T$$

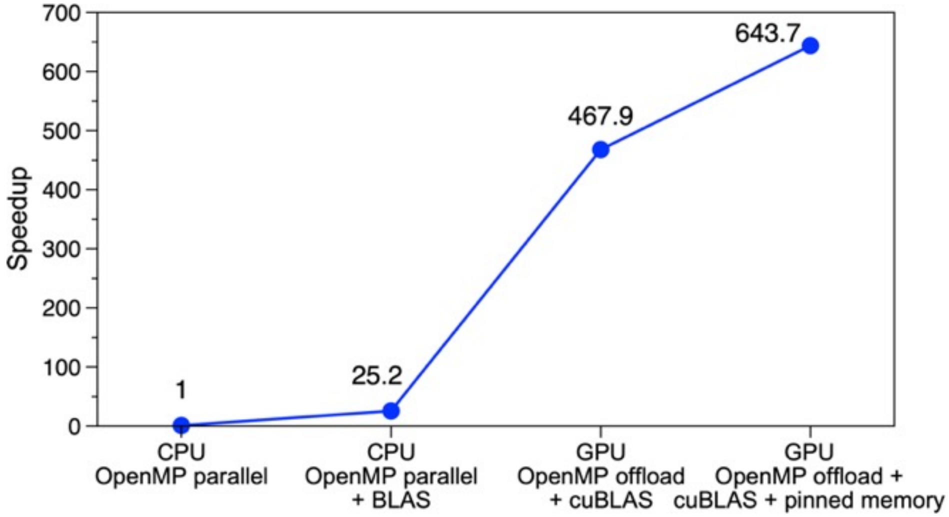
- **50.5% of the theoretical peak FLOP/s performance on 786,432 Blue Gene/Q cores (entire Mira at the Argonne Leadership Computing Facility)**
- **55% of the theoretical peak FLOP/s on Intel Xeon E5-2665**

K. Nomura *et al.*, *IEEE/ACM Supercomputing, SC14* ('14)

BLASified Local Field Dynamics (LFD)

- **BLASified nonlocal LFD on graphics processing unit (GPU): Operation of nonlocal potential is projected onto *a vector space spanned by Kohn-Sham orbitals* at time 0 within the real-time scissor approximation** [Wang *et al*., *J. Phys. Condens. Mat*. **31**, 214002 ('19)]**, making it dense matrix operations implemented with highly optimized level3 (or matrix-matrix) BLAS (basic linear algebra subprogram) library on GPU**

$$\hat{v}_{\mathrm{nl}}|\psi_n(t)\rangle \cong \Delta_{\mathrm{sci}} \sum_{m\geq\mathrm{LUMO}} |\psi_m\rangle\langle\psi_m|\psi_n(t)\rangle$$

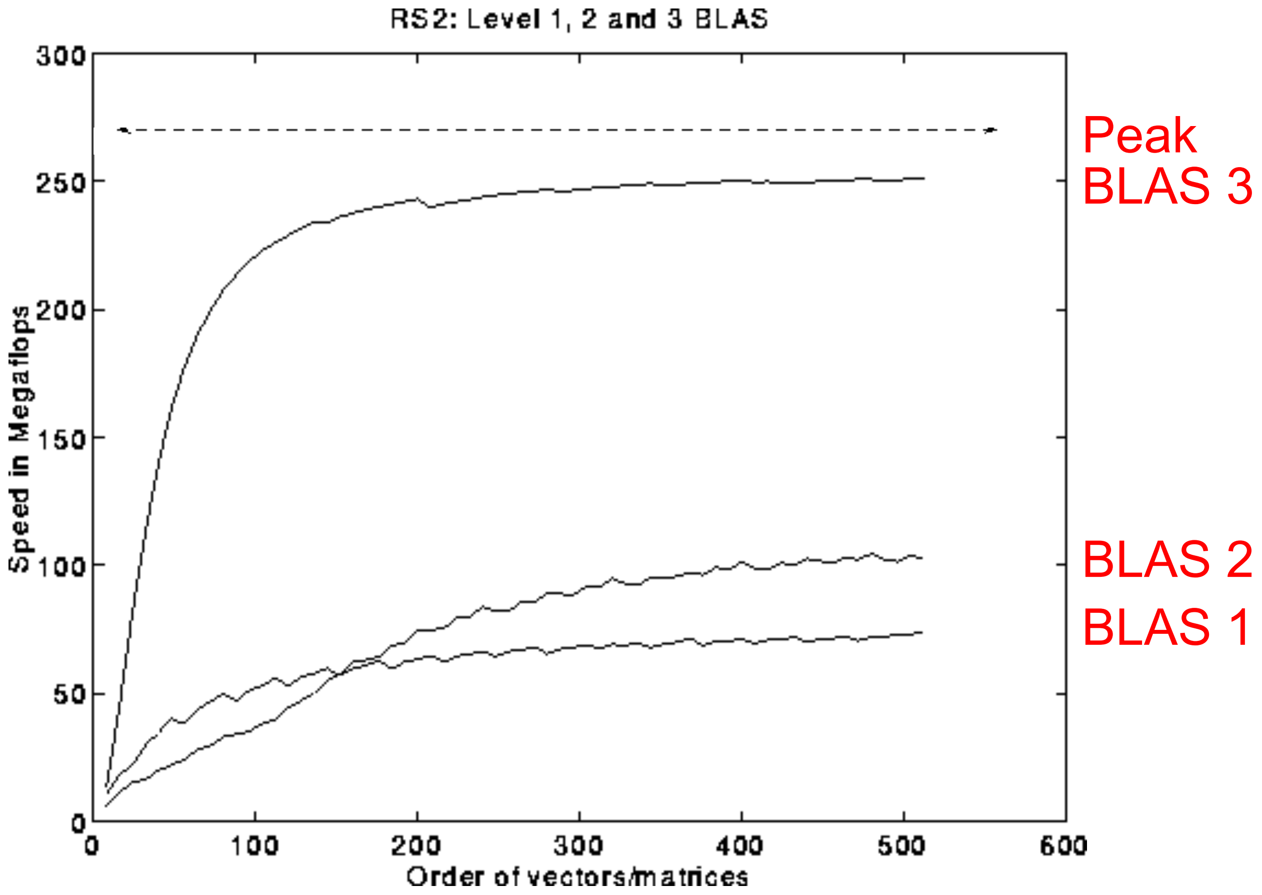


- **See lecture on** divide-&conquer Maxwell-Ehrenfest-surface hopping simulation

Razakh *et al*., *PDSEC* (IEEE, '24); Piroozan *et al*., *PMBS* (IEEE, '24)

BLAS3-Performance Molecular Dynamics?

- **BLAS3: q = flop/memory access = (block size)$^{1/2}$**



- **Molecular dynamics: $q = O(n^2)/O(n) = O(n\text{: block size})$**

Quantum MD@Scale

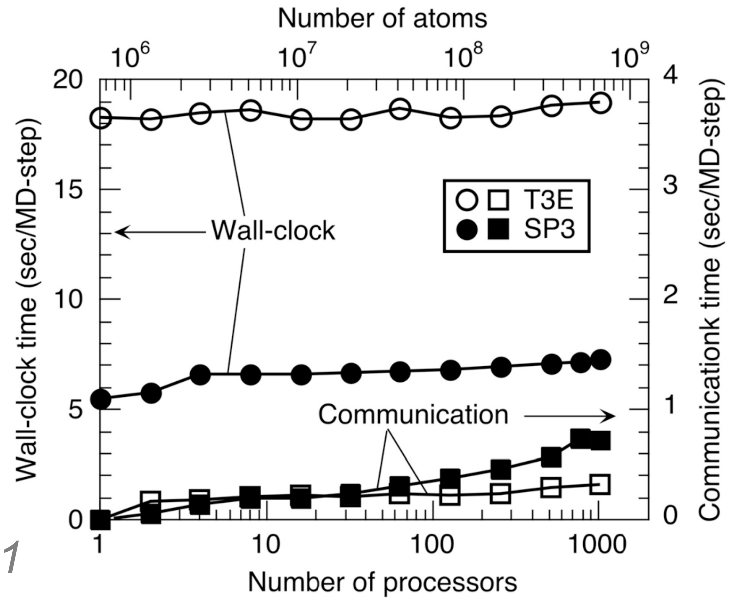
Quantum dynamics at scale: ultrafast control of emergent functional materials

S. C. Tiwari, P. Sakdhnagool, R. K. Kalia, A. Krishnamoorthy, M. Kunaseth,
A. Nakano, K. Nomura, P. Rajak, F. Shimojo, Y. Luo & P. Vashishta

Best Paper in *ACM HPC Asia 2020*

Scalable atomistic simulation algorithms for materials research, A. Nakano *et al*.,
Best Paper, *IEEE/ACM Supercomputing 2001, SC01*

19 years since

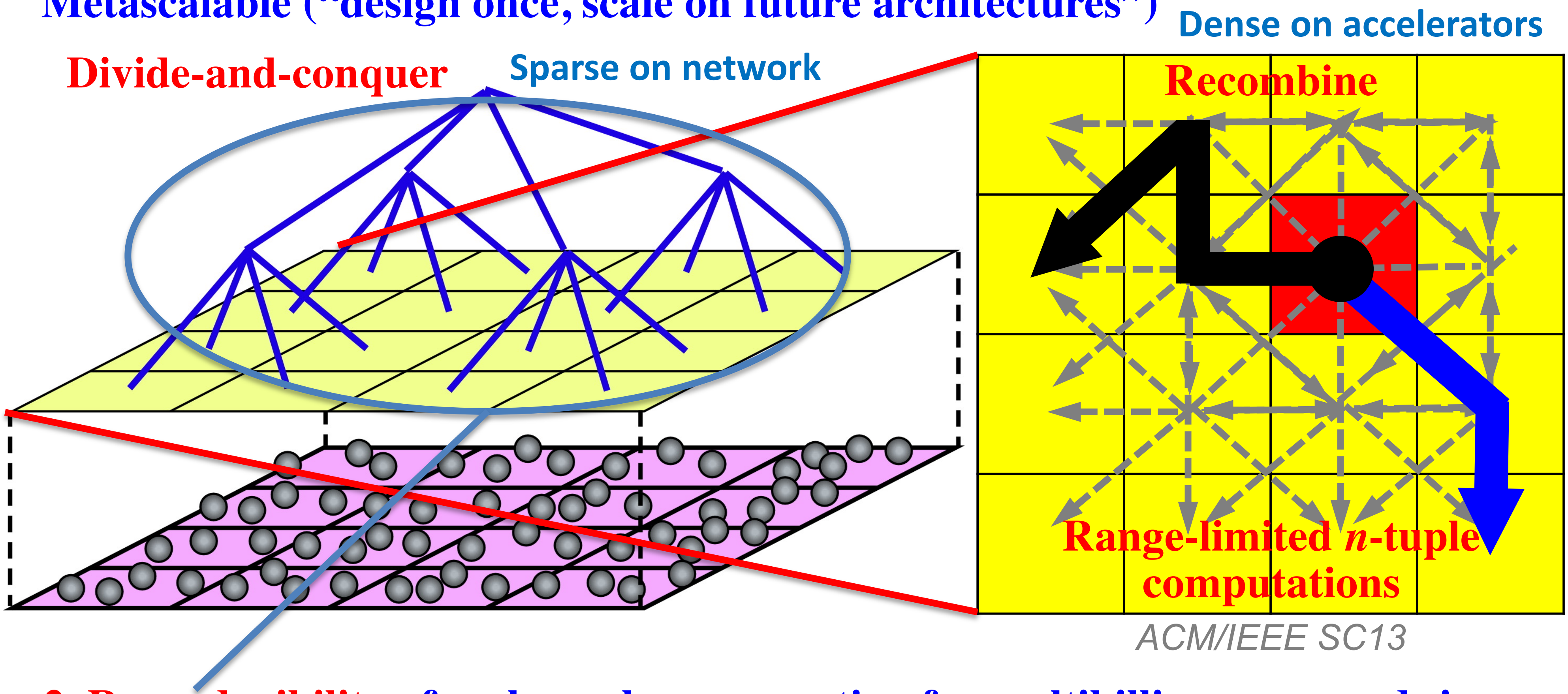


Exascale Computing Challenges

1. **Scalability for billion-way parallelism**

J. Chem. Phys. **140**, 18A529 ('14)
IEEE/ACM SC14
IEEE Computer **48(11)**, 33 ('15)

Divide-conquer-recombine (DCR) algorithmic framework
Metascalable ("design once, scale on future architectures")



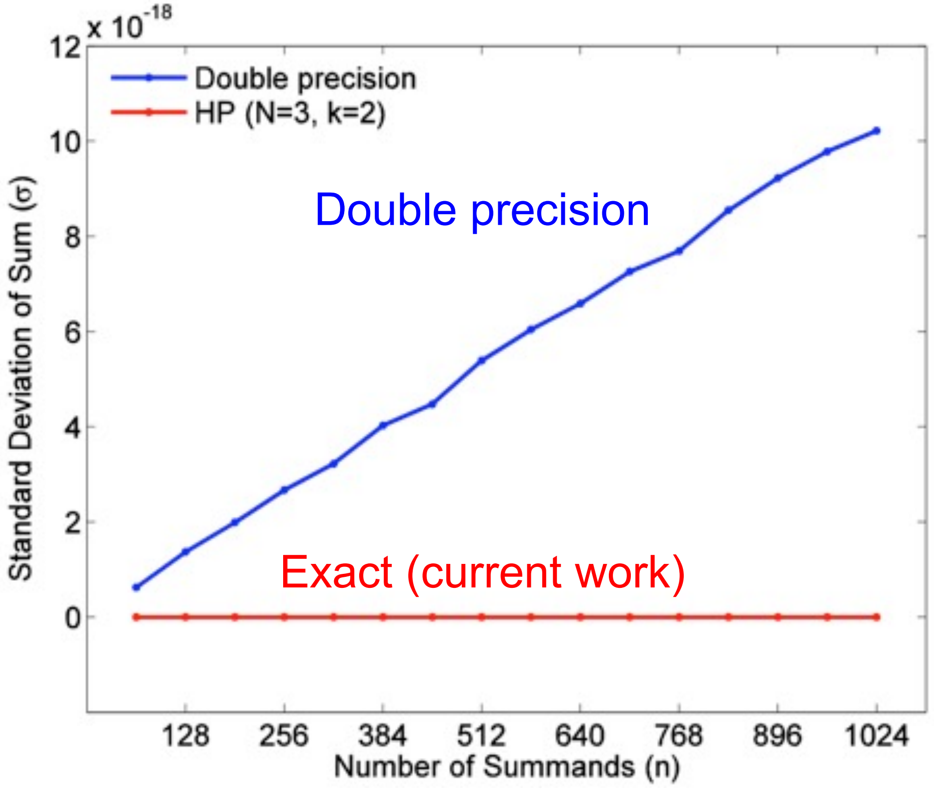
ACM/IEEE SC13

2. **Reproducibility of real-number summation for multibillion summands in the global sum; double-precision arithmetic began to produce different results on different high-end architectures**

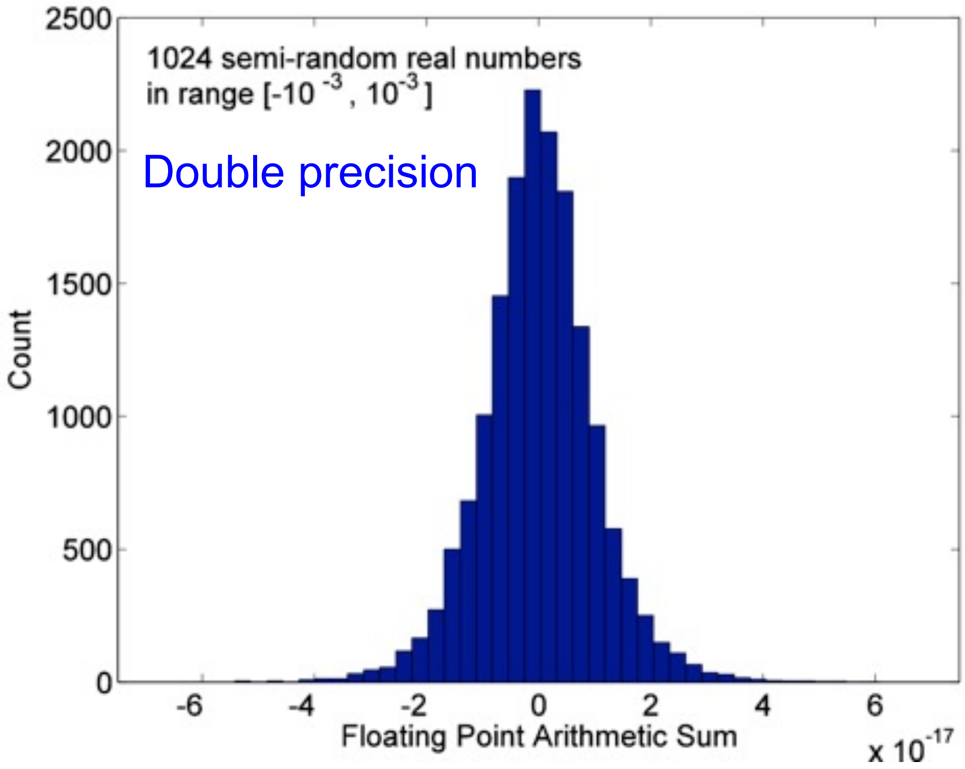
Reproducibility Challenge

- **Rounding (truncation) error makes floating-point addition non-associative**

$$(a + b) + c \neq a + (b + c)$$



Standard deviation of sum with random summation orders



Distribution of sum with random summation orders

- **Finding: Sum becomes a random walk across the space of possible rounding error**

High-Precision (HP) Method

- **Propose an extension of the order-invariant, higher-precision intermediate-sum method by Hallberg & Adcroft** [*Par. Comput*. **40**, 140 ('14)]
- **The proposed variation represents a real number *r* using a set of *N* 64-bit unsigned integers, a_i (i = 0, *N*-1)**

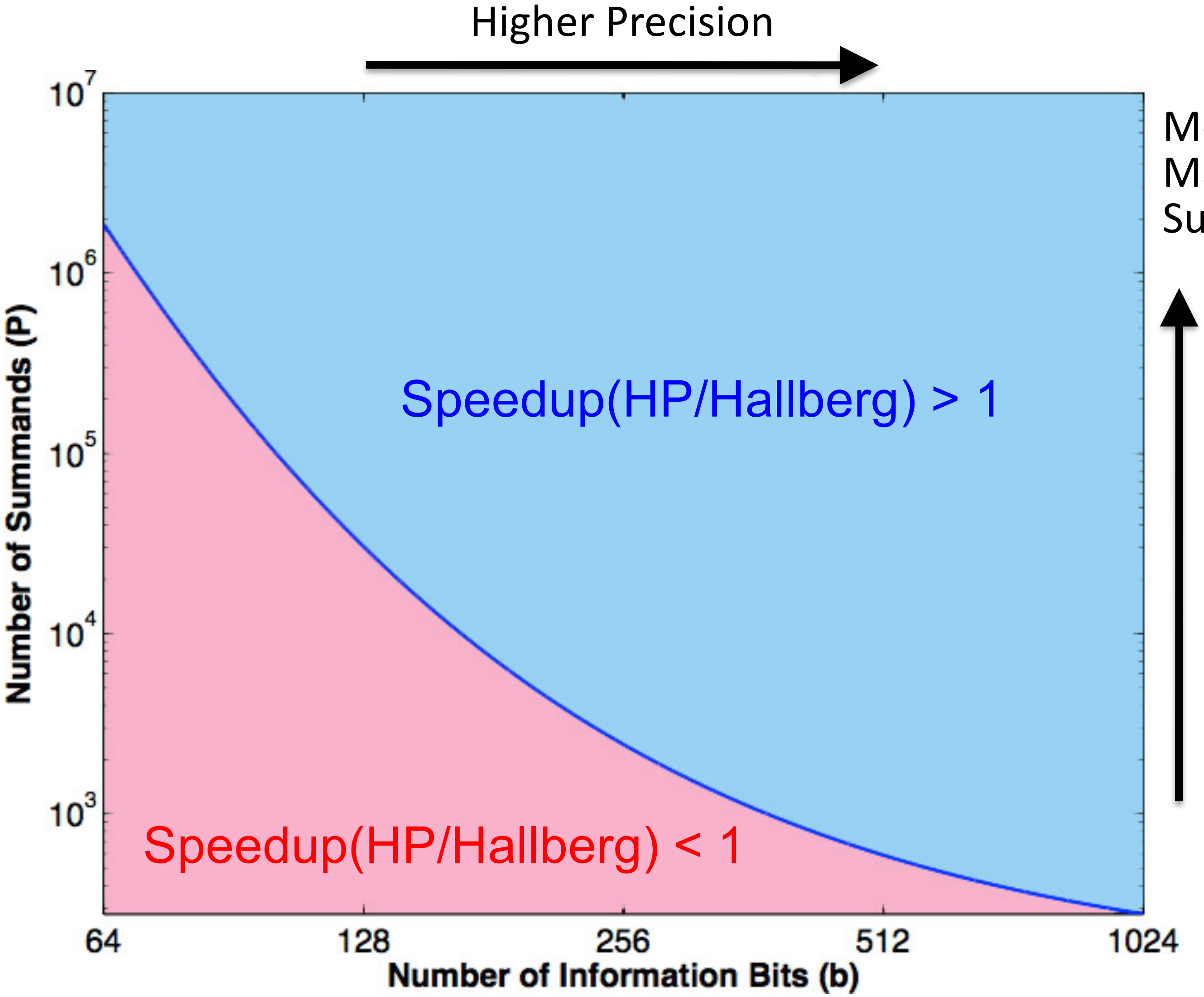
$$r = \sum_{i=0}^{N-1} a_i 2^{64(N-k-i-1)}$$
$$= \overbrace{a_0 2^{64(N-k-1)} + \cdots + a_{N-k-1}}^{N-k} + \cdots + \overbrace{a_{N-k} 2^{-64} + \cdots + a_{N-1} 2^{-64k}}^{k}$$

- ***k* is the number of 64-bit unsigned integers assigned to represent the fractional portion of *r* (0 ≤ *k* ≤ *N*), whereas *N*−*k* integers represent the whole-number component**
- **Negative number is represented by two's complement in integer representation, using only 1 bit**

P. E. Small *et al.*, *IEEE IPDPS 2016*

Performance Projection

- **HP sum is faster than Hallberg sum for higher precision & larger numbers of summands**



More Massive Sum

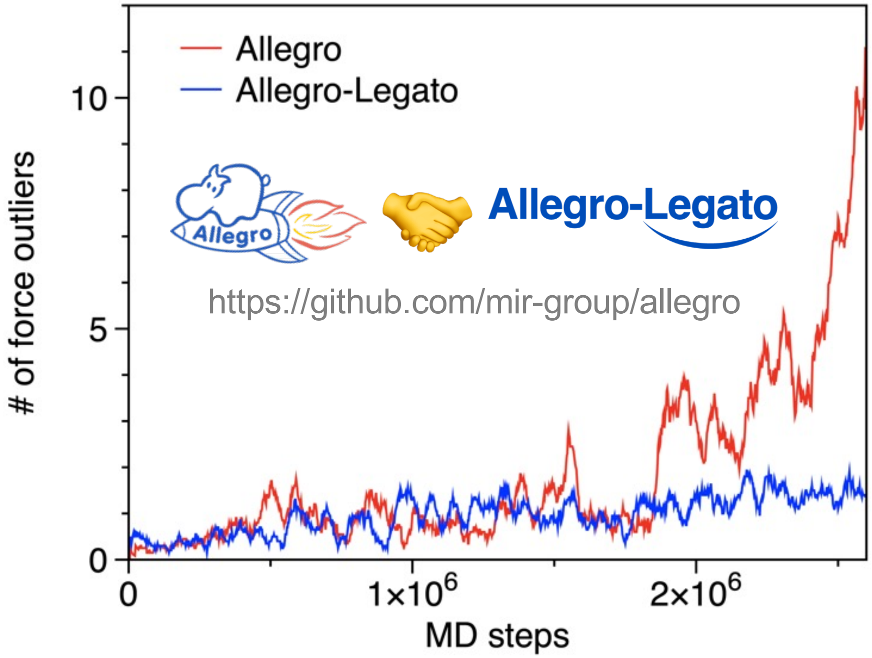
P. E. Small *et al.*, *IEEE IPDPS 2016*

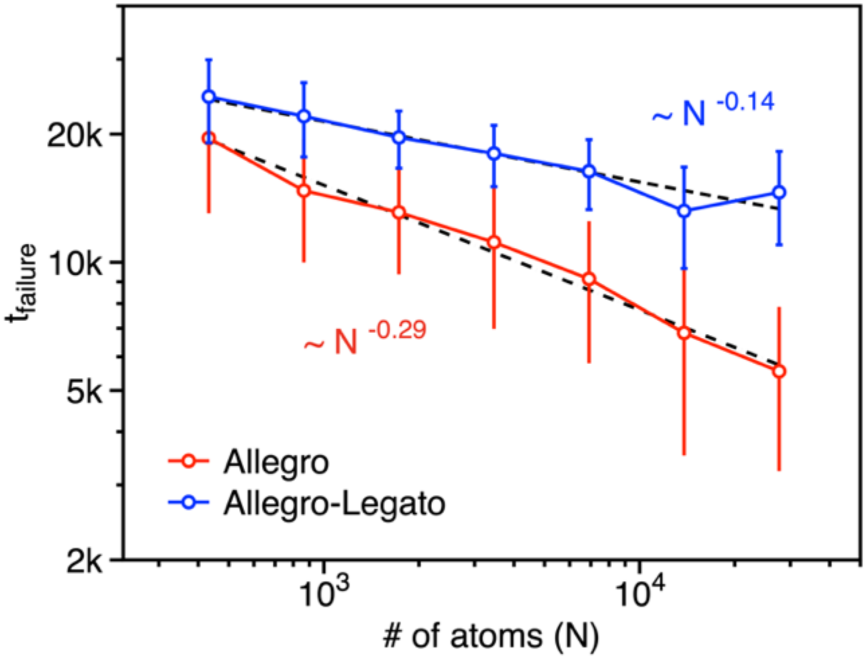
New: Fidelity Scaling Challenge

- **Allegro (fast) NNQMD: State-of-the-art *accuracy & speed* founded on group-theoretical equivariance & local descriptors** [Musaelian *et al*., *Nat. Commun.* **14**, 579 ('23)]
- **Fidelity-scaling problem: On massively parallel computers, growing number of unphysical (adversarial) force predictions prohibits simulations involving larger numbers of atoms for longer times**

NNQMD: neural-network quantum molecular dynamics

- **Allegro-Legato (fast and "smooth"): *Sharpness aware minimization (SAM)* enhances the *robustness* of Allegro through improved smoothness of loss landscape**
  $\mathbf{w}_* = \mathrm{argmin}_{\mathbf{w}}\left[L(\mathbf{w}) + \max_{\|\boldsymbol{\epsilon}\|_2 \le \rho}\{L(\mathbf{w}+\boldsymbol{\epsilon}) - L(\mathbf{w})\}\right]$ (L: loss; $\mathbf{w}$: model parameters)
- **Elongated time-to-failure scaling, $t_{\text{failure}} = O(N^{-\beta})$, without sacrificing accuracy or speed, thereby achieving spectroscopically stable long-time Hamiltonian trajectory**





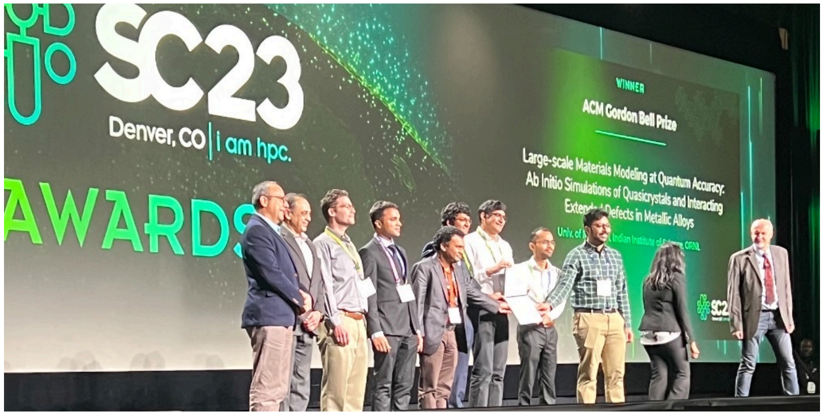
H. Ibayashi *et al*., *ISC23—LNCS* **13948**, 223 ('23); *arXiv*: 2303.08169

Gordon Bell Prizes

aka Nobel prize of supercomputing

Gordon Bell winners in extreme-scale quantum simulations

- F. Gygi *et al*., "Large-scale electronic structure calculations of high-Z metals on the BlueGene/L platform" ('06)
- M. Eisenbach *et al*., "A scalable method for ab initio computation of free energies in nanoscale systems" ('09)
- Y. Hasegawa *et al*., "First-principles calculations of electron states of a silicon nanowire with 100,000 atoms on the K computer" ('11)
- A. N. Ziogas *et al*., "A data-centric approach to extreme-scale ab initio dissipative quantum transport simulations" ('19)
- S. Das *et al*., "Large-scale materials modeling at quantum accuracy: *Ab initio* simulations of quasicrystals and interacting extended defects in metallic alloys" ('23)


SC23
Denver, CO | i am hpc.
AWARDS
WINNER
ACM Gordon Bell Prize
Large-scale Materials Modeling at Quantum Accuracy:
Ab Initio Simulations of Quasicrystals and Interacting
Extended Defects in Metallic Alloys

See the reading list

Where to Go from Here: Performance

- **Performance profiling: First thing to find is how well/badly your program is performing in terms of flop/s performance, vectorization, cache miss, *etc*.**
- **Use professional tools like Intel VTune & Advisor if available on your computer:**

https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html
https://www.intel.com/content/www/us/en/developer/tools/oneapi/advisor.html
https://www.youtube.com/watch?reload=9&v=ymy139CuAx8

Advisor can show you the "roofline" of your application

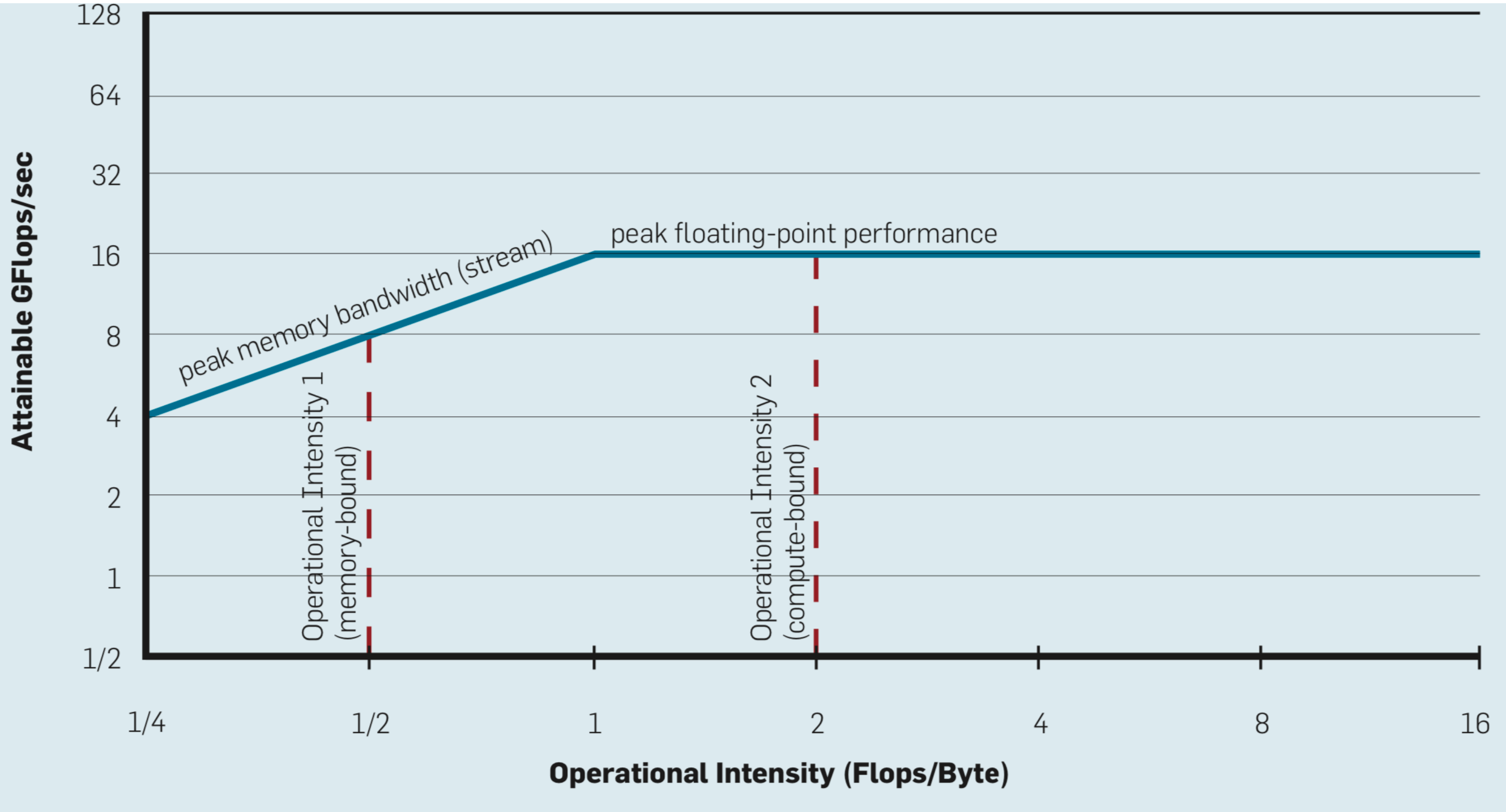
- **Off-chip memory bandwidth (from DRAM) is critical for performance (to feed enough data to be operated)**
- ***Operational intensity*: Operations per byte of DRAM traffic**
- ***Roofline model*: Predicts the floating-point (fp) performance from operation intensity, theoretical peak fp performance & peak memory bandwidth**

$$Attainable fp\ performance \left[\frac{\mathrm{Gflop}}{\mathrm{sec}}\right] = \min\begin{pmatrix} Peak\ fp\ performance \left[\frac{\mathrm{Gflop}}{\mathrm{sec}}\right], \\ Peak\ memory\ bandwidth \left[\frac{\mathrm{GByte}}{\mathrm{sec}}\right] \times Operational\ intensity \left[\frac{\mathrm{flop}}{\mathrm{Byte}}\right] \end{pmatrix}$$

S. Williams *et al*., *Commun. ACM* 52(4), 65 ('09)
V. Elango *et al*., *ACM. T. Arch. Code Opt.* 11, 67 ('15)

Roofline Model of Performance

$$Attainable\ fp\left[\frac{\text{Gflop}}{\text{sec}}\right] =$$
$$\min\left(Peak\ fp\left[\frac{\text{Gflop}}{\text{sec}}\right], Memory\ bandwidth\left[\frac{\text{GByte}}{\text{sec}}\right] \times Operational\ intensity\ \left[\frac{\text{flop}}{\text{Byte}}\right]\right)$$



Key: Data/computation locality

See lecture on performance optimization for performance-optimization techniques