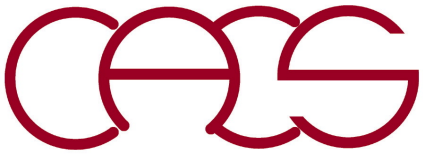
Divide-&-Conquer Parallelism

Aiichiro Nakano

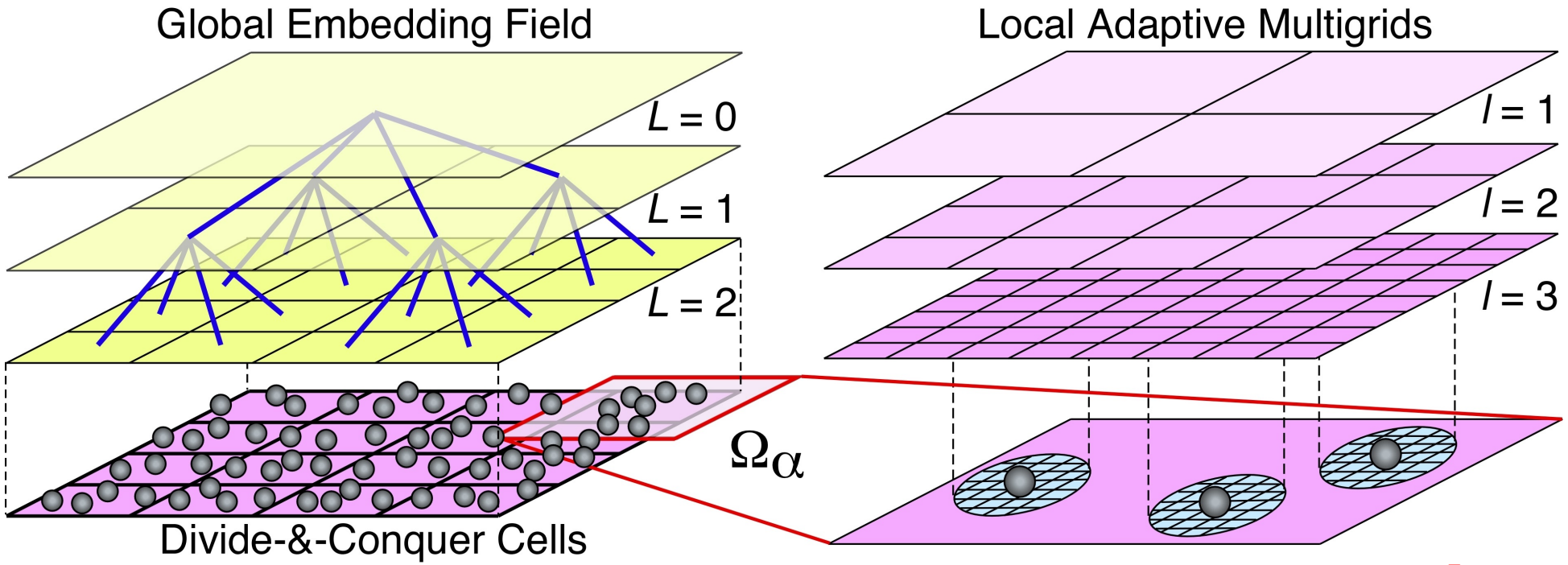
Collaboratory for Advanced Computing & Simulations
Department of Computer Science
Department of Physics & Astronomy
Department of Quantitative & Computational Biology
University of Southern California

Email: anakano@usc.edu

Goal: How to program D-&-C using MPI?
Communicator!



Embedded Divide-&-Conquer Algorithms



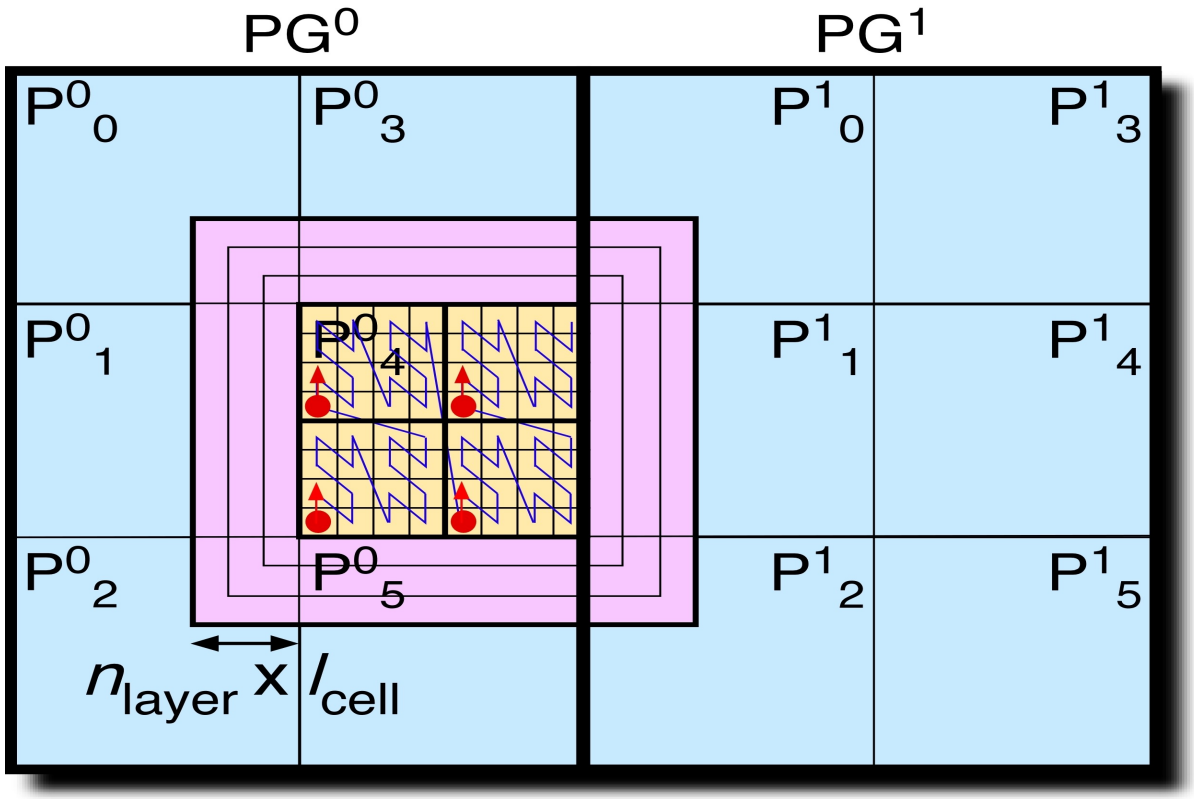
D-&-C for: (1) $O(N)$ algorithms; & (2) scalability > $P = 10^5$

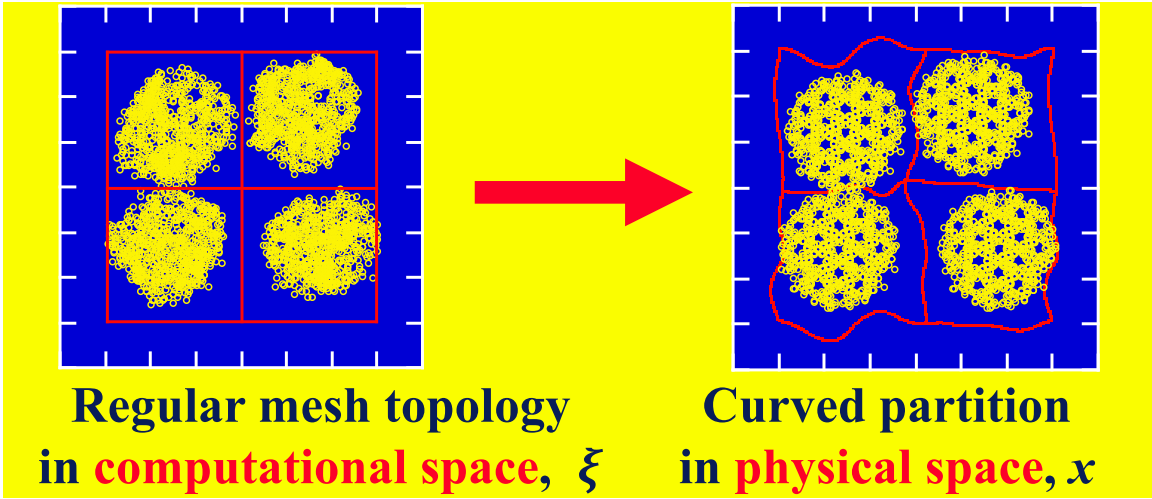
- ***N*-body problem: $O(N^2) \rightarrow O(N)$**
  - > **Space-time multiresolution molecular dynamics (MRMD):**
    Fast multipole method & symplectic multiple time stepping
- **Variable *N*-charge problem: $O(N^3) \rightarrow O(N)$**
  - > **Fast reactive force-field (F-ReaxFF) MD: Multilevel preconditioning**
- **Quantum *N*-body problem: $O(C^N) \rightarrow O(N)$**
  - > **EDC density functional theory (EDC-DFT): Adaptive multigrids**

Scalable Parallelization Framework

HCD: tunable hierarchical cellular decomposition

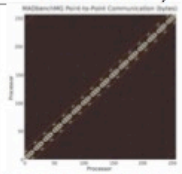
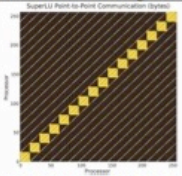
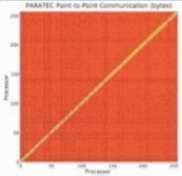
- Topology-preserving computational-space decomposition
- Computational cell ⊂ thread ⊂ processor ⊂ processor group
- AI-based computation/ data layout tuning
- Wavelet-based adaptive load balancing
- Spacefilling-curve data compression for I/O

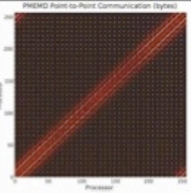
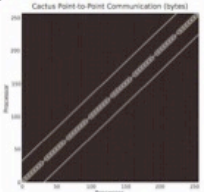




The Landscape of Parallel Computing Research: A View from Berkeley

7 dwarfs (dwarf = algorithmic method that captures a pattern of computation & communication) + 6 combinatorial dwarfs

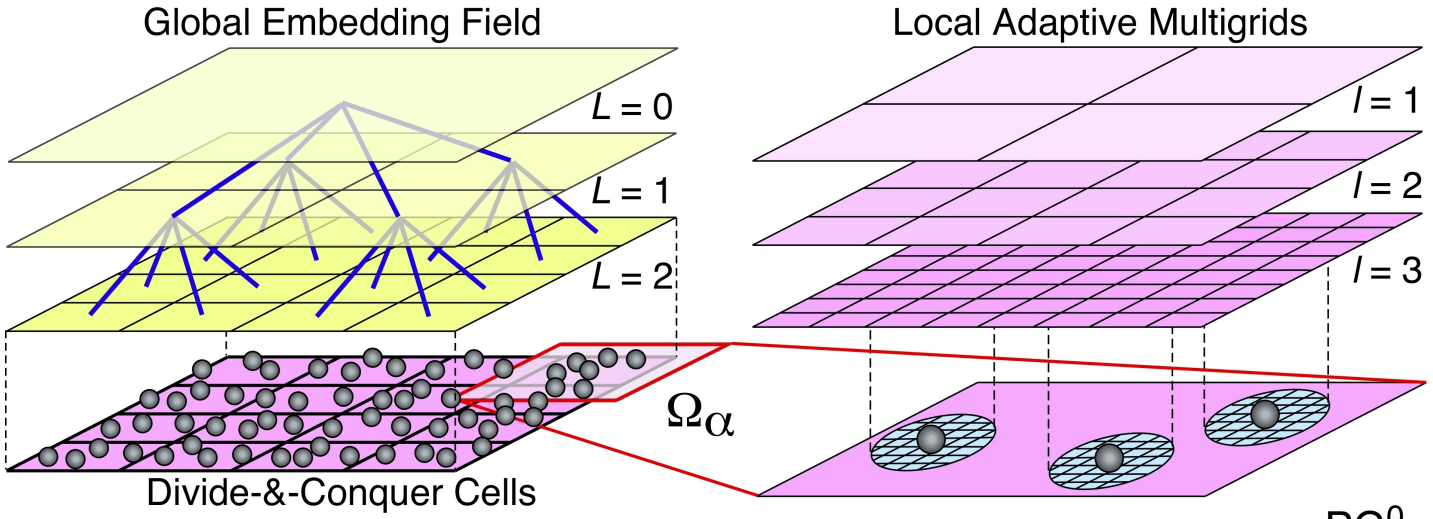
Dwarf	*Description*	*Communication Pattern (Figure axes show processors 1 to 256, with black meaning no communication)*	*NAS Benchmark / Example HW*
1. Dense Linear Algebra (e.g., BLAS [Blackford et al 2002], ScaLAPACK [Blackford et al 1996], or MATLAB [MathWorks 2006])	Data are dense matrices or vectors. (BLAS Level 1 = vector-vector; Level 2 = matrix-vector; and Level 3 = matrix-matrix.) Generally, such applications use unit-stride memory accesses to read data from rows, and strided accesses to read data from columns.	 The communication pattern of MadBench, which makes heavy use of ScaLAPACK for parallel dense linear algebra, is typical of a much broader class of numerical algorithms	Block Triadiagonal Matrix, Lower Upper Symmetric Gauss-Seidel / Vector computers, Array computers
2. Sparse Linear Algebra (e.g., SpMV, OSKI [OSKI 2006], or SuperLU [Demmel et al 1999])	Data sets include many zero values. Data is usually stored in compressed matrices to reduce the storage and bandwidth requirements to access all of the nonzero values. One example is block compressed sparse row (BCSR). Because of the compressed formats, data is generally accessed with indexed loads and stores.	 SuperLU (communication pattern pictured above) uses the BCSR method for implementing sparse LU factorization.	Conjugate Gradient / Vector computers with gather/scatter
3. Spectral Methods (e.g., FFT [Cooley and Tukey 1965])	Data are in the frequency domain, as opposed to time or spatial domains. Typically, spectral methods use multiple butterfly stages, which combine multiply-add operations and a specific pattern of data permutation, with all-to-all communication for some stages and strictly local for others.	 PARATEC: The 3D FFT requires an all-to-all communication to implement a 3D transpose, which requires communication between every link. The diagonal stripe describes BLAS-3 dominated linear-algebra step required for orthogonalization.	Fourier Transform / DSPs, Zalink PDSP [Zarlink 2006]

Dwarf	*Description*	*Communication Pattern (Figure axes show processors 1 to 256, with black meaning no communication)*	*NAS Benchmark / Example HW*
4. N-Body Methods (e.g., Barnes-Hut [Barnes and Hut 1986], Fast Multipole Method [Greengard and Rokhlin 1987])	Depends on interactions between many discrete points. Variations include particle-particle methods, where every point depends on all others, leading to an $O(N^2)$ calculation, and hierarchical particle methods, which combine forces or potentials from multiple points to reduce the computational complexity to O(N log N) or O(N).	 PMEMD's communication pattern is that of a particle mesh Ewald calculation.	(no benchmark) / GRAPE [Tokyo 2006], MD-GRAPE [IBM 2006]
5. Structured Grids (e.g., Cactus [Goodale et al 2003] or Lattice-Boltzmann Magneto-hydrodynamics [LBMHD 2005])	Represented by a regular grid; points on grid are conceptually updated together. It has high spatial locality. Updates may be in place or between 2 versions of the grid. The grid may be subdivided into finer grids in areas of interest ("Adaptive Mesh Refinement"); and the transition between granularities may happen dynamically.	 Communication pattern for Cactus, a PDE solver using 7-point stencil on 3D block-structured grids.	Multi-Grid, Scalar Penta-diagonal / QCDOC [Edinburg 2006], BlueGeneL
6. Unstructured Grids (e.g., ABAQUS [ABAQUS 2006] or FIDAP [FLUENT 2006])	An irregular grid where data locations are selected, usually by underlying characteristics of the application. Data point location and connectivity of neighboring points must be explicit. The points on the grid are conceptually updated together. Updates typically involve multiple levels of memory reference indirection, as an update to any point requires first determining a list of neighboring points, and then loading values from those neighboring points.		Unstructured Adaptive / Vector computers with gather/scatter, Tera Multi Threaded Architecture [Berry et al 2006]
7. Monte Carlo (e.g., Quantum Monte Carlo [Aspuru-Guzik et al 2005])	Calculations depend on statistical results of repeated random trials. Considered embarrassingly parallel.	Communication is typically not dominant in Monte Carlo methods.	Embarrassingly Parallel / NSF Teragrid

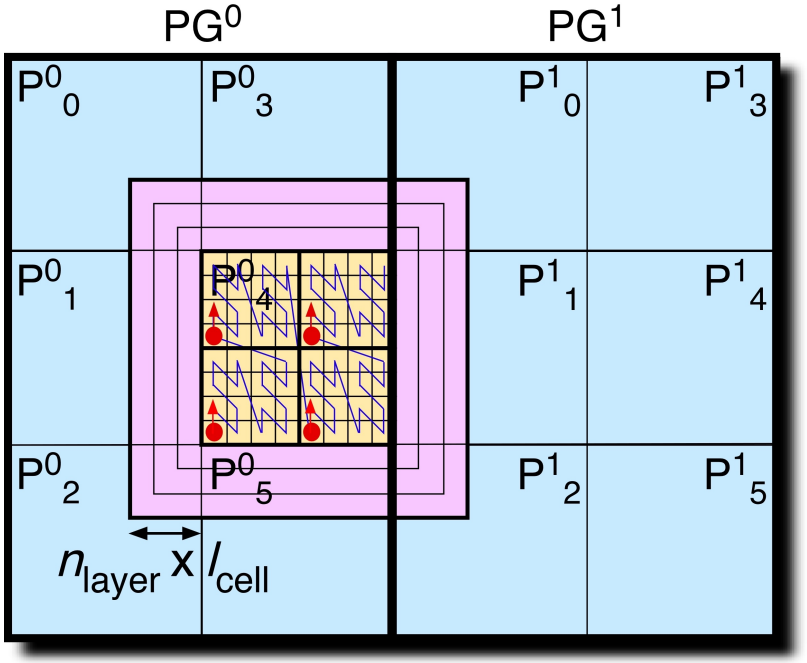
http://view.eecs.berkeley.edu/wiki/Main_Page

A Metascalable Dwarf

A metascalable (or "design once, scale on new architectures") parallel computing framework for broad applications (*e.g.*, equation solvers, constrained optimization, search, visualization & graphs)



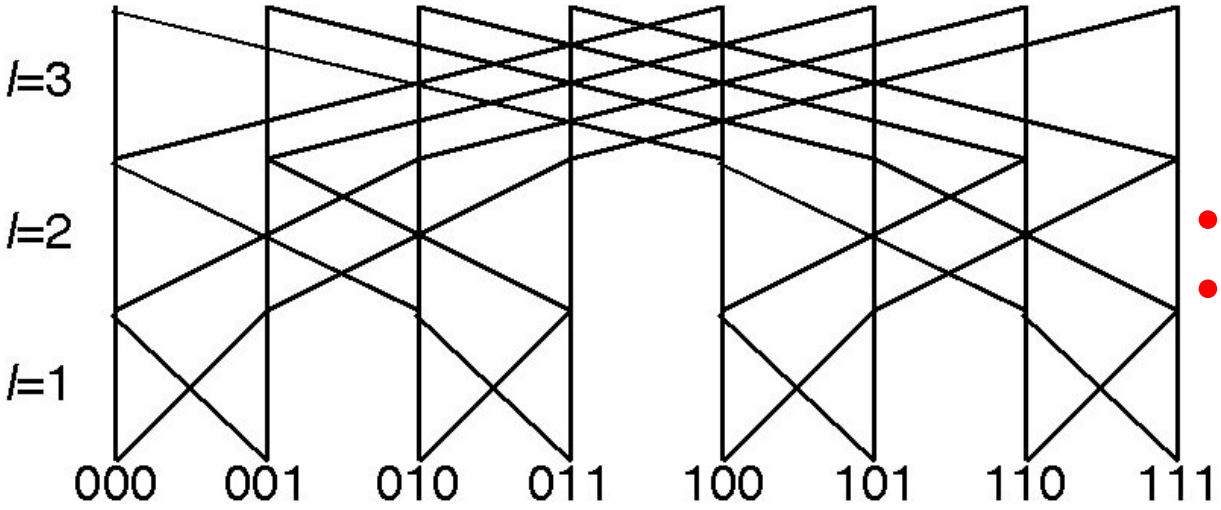
- **Embedded divide-&-conquer (EDC) algorithmic framework to design linear-scaling algorithms**
  - **Tunable hierarchical cellular decomposition (HCD) parallelization framework to map the scalable algorithms onto hardware**



K. Nomura *et al.*, *IPDPS 2009*; F. Shimojo *et al.*, *J. Phys.: Condens. Matter* **20**, 294204 ('08)

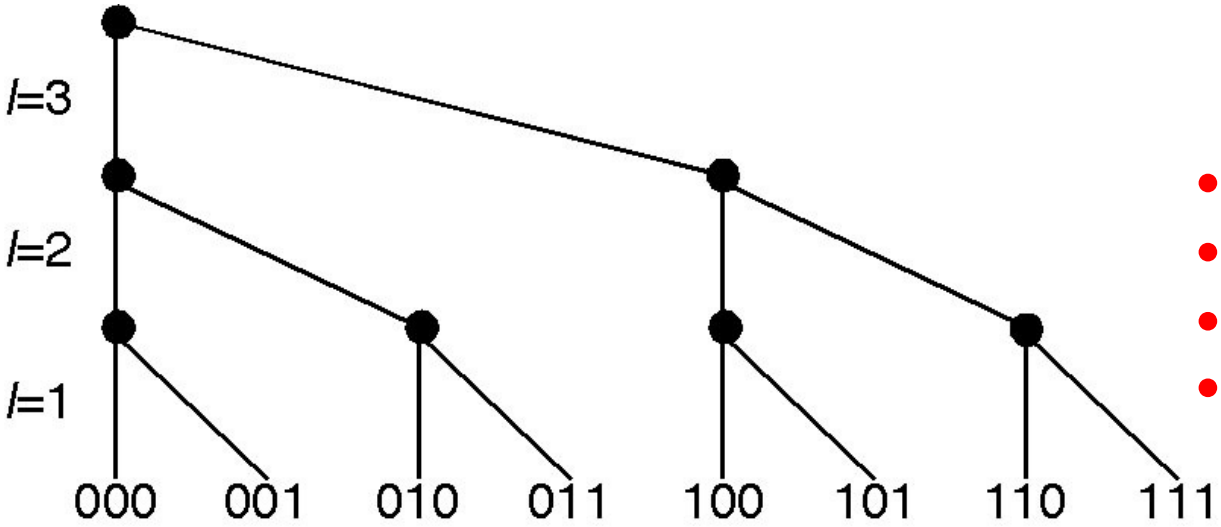
Global Communications

All-to-all (hypercube): $O(N \log N)$



- Quicksort
- Fast Fourier transform

All-to-one (tournament): $O(N)$



- Global reduction
- Fast multipole method
- Multigrid method
- Wavelets

Multigrid Method

- **Residual equation:** $\mathbf{A}^{(l)}(\mathbf{v}+\mathbf{e}) = -4\pi e^2\mathbf{n}$ (error)

$$\mathbf{A}^{(l)}\mathbf{v} = -4\pi e^2\mathbf{n} + \mathbf{r} \quad \text{(residual)}$$

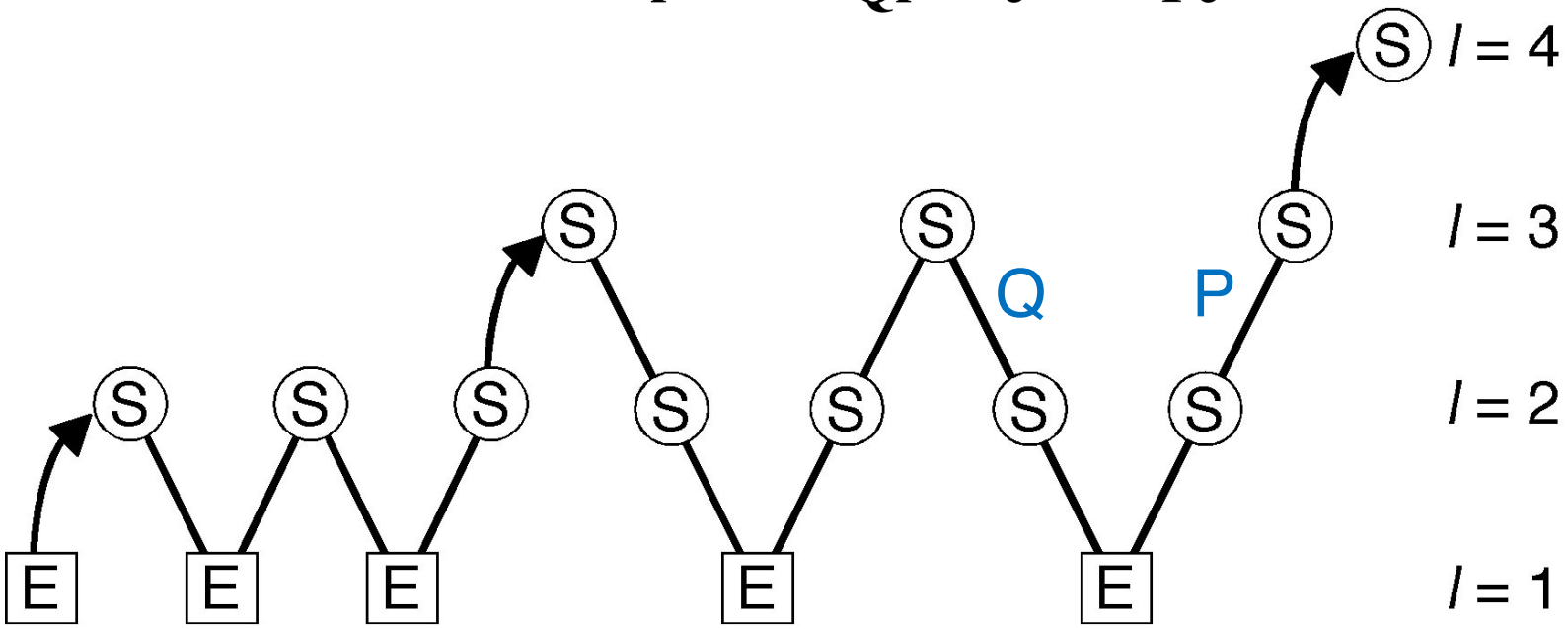
$$\mathbf{A}^{(l)}\mathbf{e} = -\mathbf{r}$$

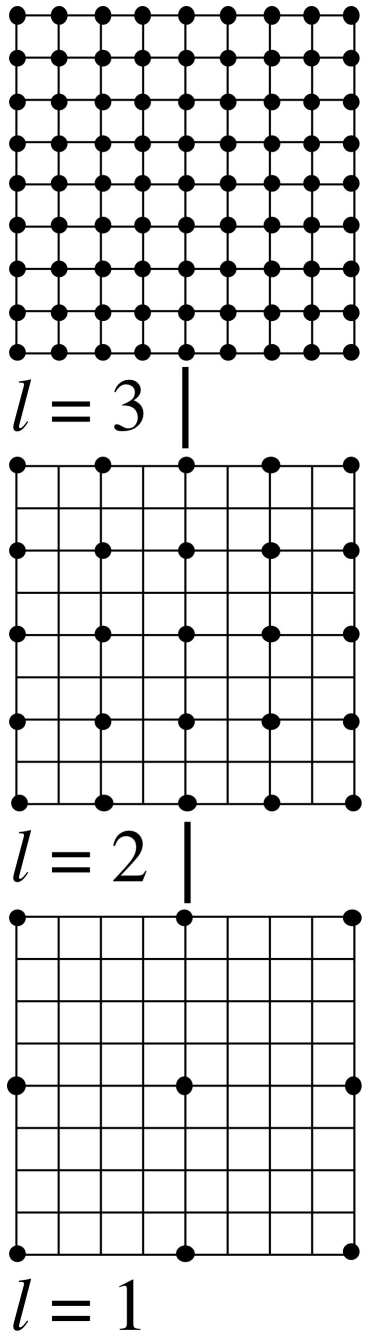
- **Smoothing (fixed-point iteration)**

$$\mathbf{e} \leftarrow \left[1 + \mathbf{Z}^{(l)}\mathbf{A}^{(l)}\right]\mathbf{e} + \mathbf{Z}^{(l)}\mathbf{r}$$

- **Coarsening (restriction) of residual & interpolation of error**

$$\mathbf{r}^{(l-1)} \leftarrow \mathbf{Q}\mathbf{r}^{(l)} \quad \mathbf{e}^{(l)} \leftarrow \mathbf{P}\mathbf{e}^{(l-1)}$$





Smoothing Example

$$(\mathbf{D}+\mathbf{L}+\mathbf{U})\mathbf{e}=-\mathbf{r}$$
$$\mathbf{D}\mathbf{e}=-(\mathbf{L}+\mathbf{U})\mathbf{e}-\mathbf{r}$$
$$\mathbf{e}=-\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{e}-\mathbf{D}^{-1}\mathbf{r}$$

↓

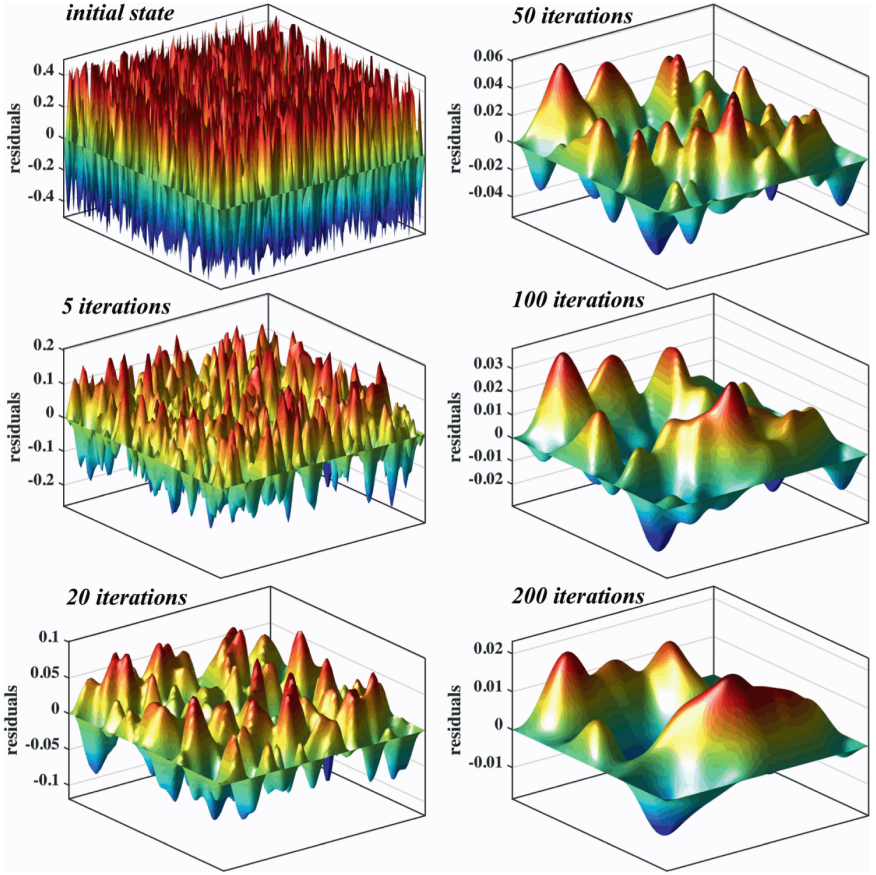
Fixed-point iteration:

$$\mathbf{e}_{\text{new}} \leftarrow -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{e}_{\text{old}}-\mathbf{D}^{-1}\mathbf{r}$$

$$\mathbf{A}=\overbrace{\begin{bmatrix}\blacksquare & & \\ & \blacksquare & \\ & & \blacksquare\end{bmatrix}}^{\mathbf{D}}+\overbrace{\begin{bmatrix} & & \\ \blacksquare & & \\ \blacksquare & \blacksquare & \end{bmatrix}}^{\mathbf{L}}+\overbrace{\begin{bmatrix} & \blacksquare & \blacksquare \\ & & \blacksquare \\ & & \end{bmatrix}}^{\mathbf{U}}$$

Diagonal Lower-triangular Upper-triangular

- **High-frequency errors die out quickly**



Adaptive Multigrids

ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computer Physics Communications

Computer Physics Communications 167 (2005) 151–164

www.elsevier.com/locate/cpc

Embedded divide-and-conquer algorithm on hierarchical real-space grids: parallel molecular dynamics simulation based on linear-scaling density functional theory

Fuyuki Shimojo [a,b], Rajiv K. Kalia [a], Aiichiro Nakano [a,*], Priya Vashishta [a]

[a] *Collaboratory for Advanced Computing and Simulations, Department of Computer Science, Department of Physics & Astronomy, Department of Materials Science & Engineering, University of Southern California, Los Angeles, CA 90089-0242, USA*

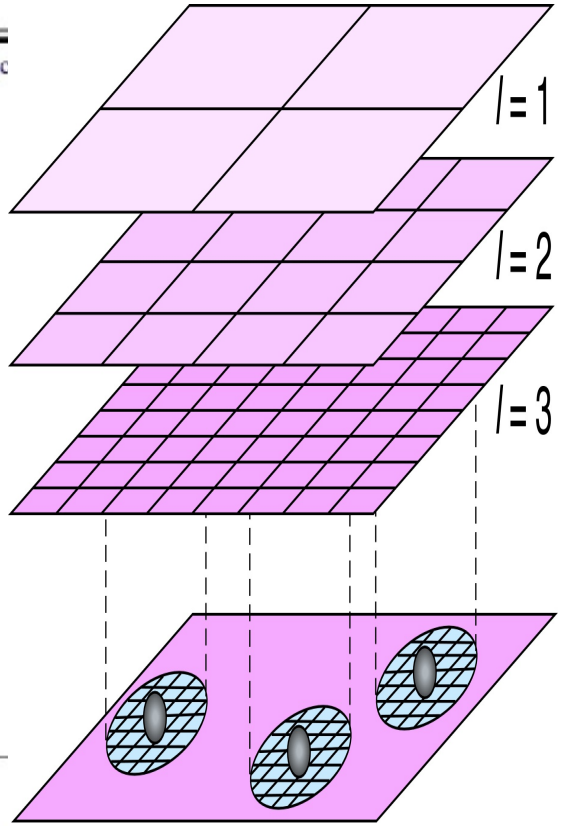
[b] *Department of Physics, Kumamoto University, Kumamoto 860-8555, Japan*

Received 5 October 2004; received in revised form 24 January 2005; accepted 26 January 2005

Available online 14 March 2005

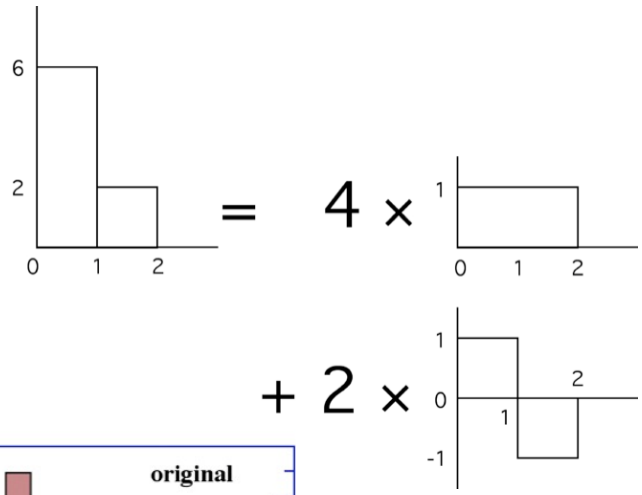
Abstract

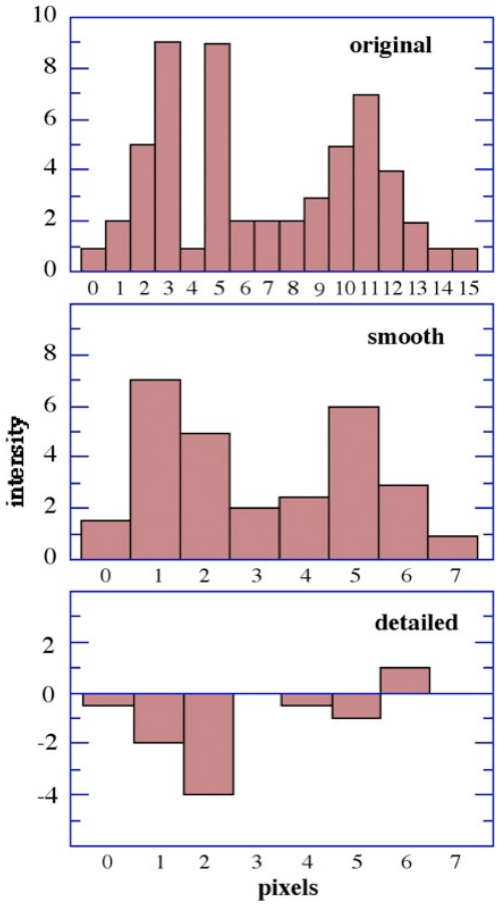
A linear-scaling algorithm has been developed to perform large-scale molecular-dynamics (MD) simulations, in which interatomic forces are computed quantum mechanically in the framework of the density functional theory. A divide-and-conquer algorithm is used to compute the electronic structure, where non-additive contribution to the kinetic energy is included with an embedded cluster scheme. Electronic wave functions are represented on a real-space grid, which is augmented with coarse multigrids to accelerate the convergence of iterative solutions and adaptive fine grids around atoms to accurately calculate ionic pseudopotentials. Spatial decomposition is employed to implement the hierarchical-grid algorithm on massively parallel computers. A converged solution to the electronic-structure problem is obtained for a 32,768-atom amorphous CdSe system on 512 IBM POWER4 processors. The total energy is well conserved during MD simulations of liquid Rb, showing the applicability of this algorithm to first principles MD simulations. The parallel efficiency is 0.985 on 128 Intel Xeon processors for a 65,536-atom CdSe system.

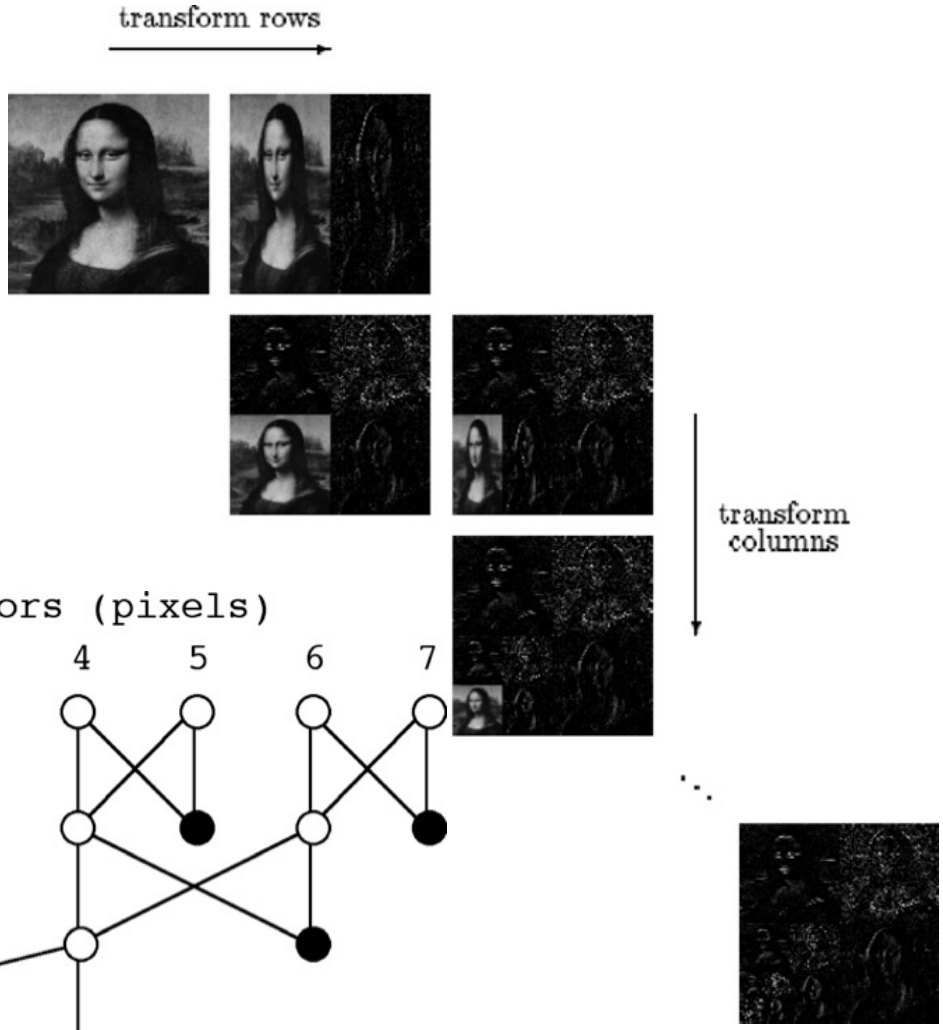


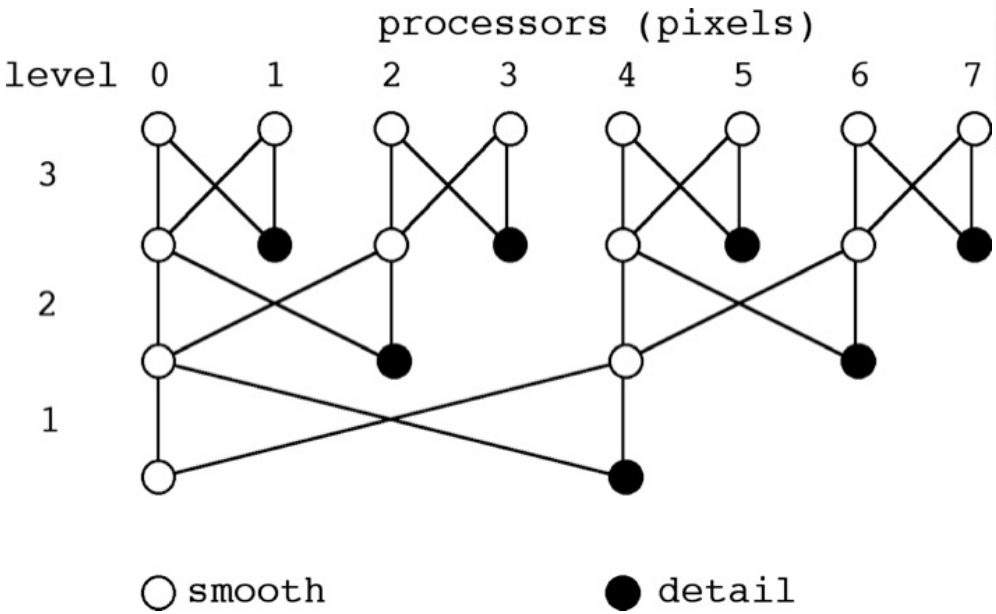
Also, Shimojo *et al.*, *Phys. Rev. B* **77**, 085103 ('08)

Wavelets









E. J. Stollnitz, T. D. DeRose, and D. H. Salesin,
IEEE Computer Graphics Appl. **15**(3), 76 ('95)

Wavelets for Model Reduction

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

Finite Elements in Analysis and Design 43 (2007) 346–360

FINITE ELEMENTS IN ANALYSIS AND DESIGN

www.elsevier.com/locate/finel

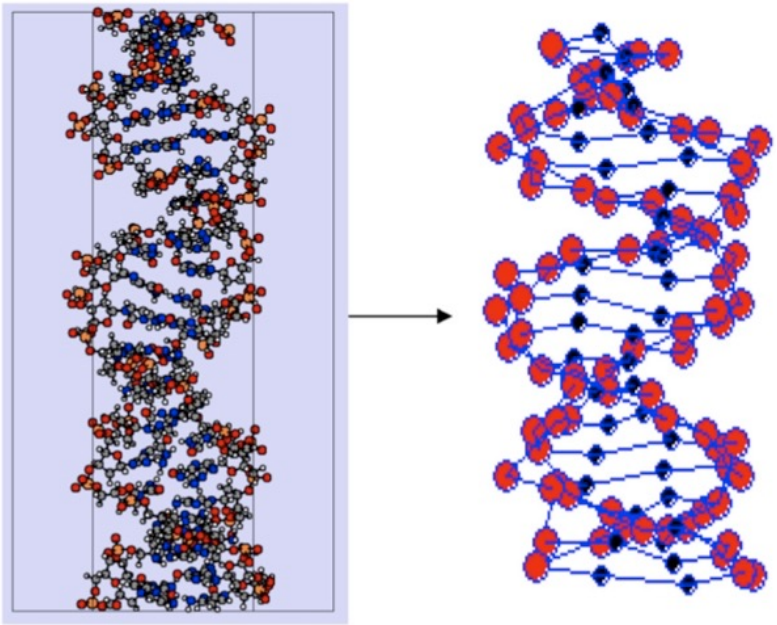
Wavelet-based multi-scale coarse graining approach for DNA molecules

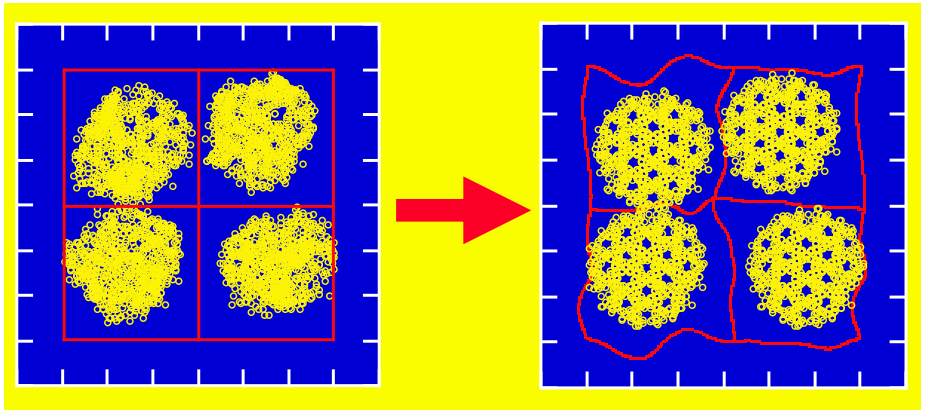
Jiun-Shyan Chen[a,*], Hailong Teng[a], Aiichiro Nakano[b]

[a]Civil and Environmental Engineering Department, University of California, Los Angeles, CA 90095-1593, USA
[b]Department of Computer Science, University of Southern California, Los Angeles, CA 90089-0242, USA

Abstract

In this work, a coarse graining technique based on a multi-scale wavelet projection is proposed for modeling of DNA molecules. Based on the fine scale atomistic response and the Henderson's theorem, the distribution functions between centers of mass of two groups of atoms are employed to obtain the fine scale potential functions. These fine scale potential functions are then homogenized using the multi-scale wavelet projection to yield the coarse-scale effective potential functions between superatoms. Molecular dynamics simulation of DNA molecules under stretching process demonstrates that the results of fine scale model and the proposed coarse grained DNA model are in good agreement. The coarse grained simulation with a significant saving of computation time is shown to be in good agreement with the fine scale simulation. Due to the reduced spatial degrees of freedom and temporal discretization, a computational efficiency of three to four orders of magnitude is achieved in the proposed coarse grained model.





CONCURRENCY: PRACTICE AND EXPERIENCE

Concurrency: Pract. Exper., Vol. 11(7), 343–353 (1999)

Multiresolution load balancing in curved space: the wavelet representation

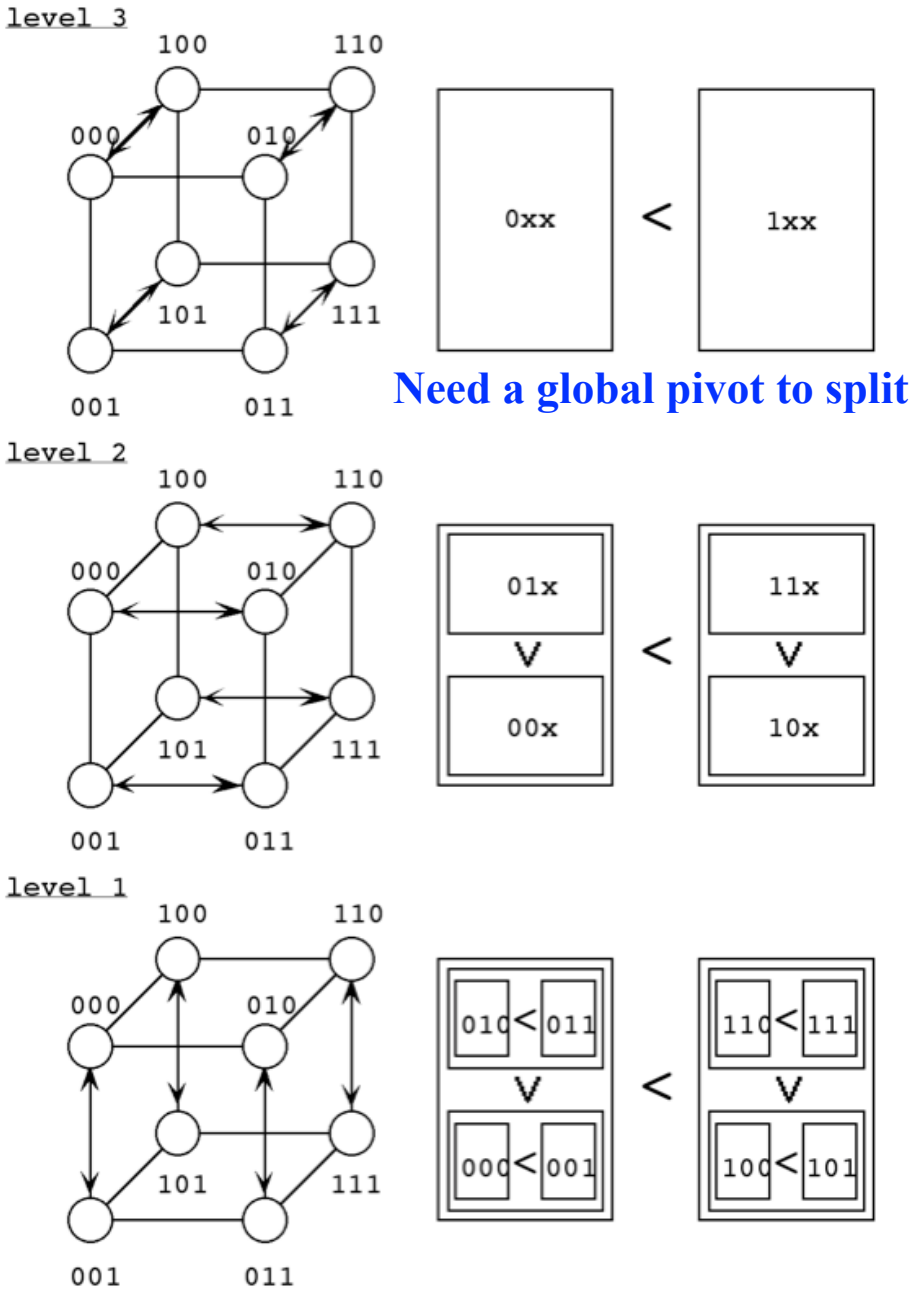
AIICHIRO NAKANO*

Department of Computer Science, Concurrent Computing Laboratory for Materials Simulations, Louisiana State University, Baton Rouge, LA 70803-4020, USA
(e-mail: nakano@bit.csc.lsu.edu)

SUMMARY

A new load-balancing scheme based on a multiresolution analysis is developed for parallel particle simulations. Workloads are partitioned with a uniform 3-dimensional mesh in an adaptive curvilinear co-ordinate system which is represented by a wavelet basis. Simulated annealing is used to determine the optimal wavelet coefficients which minimize load imbalance and communication costs. Performance tests on a parallel computer involving up to 1.04 billion particles demonstrate the scalability of the new load balancer. Copyright © 1999 John Wiley & Sons, Ltd.

Hypercube Sort



Basis: Quicksort

In putting together this issue of *Computing in Science & Engineering*, we knew three things: it would be difficult to list just 10 algorithms; it would be fun to assemble the authors and read their papers; and, whatever we came up with in the end, it would be controversial. We tried to assemble the 10 algorithms with the greatest influence on the development and practice of science and engineering in the 20th century. Following is our list (here, the list is in chronological order; however, the articles appear in no particular order):

- Metropolis Algorithm for Monte Carlo
- Simplex Method for Linear Programming
- Krylov Subspace Iteration Methods
- The Decompositional Approach to Matrix Computations
- The Fortran Optimizing Compiler
- QR Algorithm for Computing Eigenvalues
- Quicksort Algorithm for Sorting
- Fast Fourier Transform
- Integer Relation Detection
- Fast Multipole Method

PHYS 516

CSCI 653

IEEE Comput. Sci. Eng. **2(1)**, 22 ('00)

Quicksort: Divide-&-Conquer

```
quicksort(int list[],int left,int right) {
  int j;
  if (left < right) {
    j = partition(list,left,right);
    quicksort(list,left,j-1);
    quicksort(list,j+1,right);
  }
}
```

- **`partition`: Given `list[left:right]`, it first chooses the left-most element as a `pivot`; on return the `pivot` element is placed at the `j`-th position, &: i) `a[left],...,a[j-1]` are less than or equal to `a[j]`; ii) `a[j+1],...,a[right]` are greater than or equal to `a[j]`.**

0	1	2	3	4	5	6	7	8	9
[5	7	2	9	6	8	3	4	1	0]
[3	0	2	1	4]	5	[8	6	9	7]
[1	0	2]	3	[4]	5	[7	6]	8	[9]
[0]	1	[2]	3	[4]	5	[6]	7 []	8	[9]

Sequential Quicksort

```
void quicksort(int list[],int left,int right) {
  int pivot,i,j;
  int temp;

  if (left < right) {
    i = left; j = right + 1;
    pivot = list[left];
    do {
      while (list[++i] < pivot && i <= right);
      while (list[--j] > pivot);
      if (i < j) {
        temp = list[i]; list[i] = list[j]; list[j] = temp;
      }
    } while (i < j);
    temp = list[left]; list[left] = list[j]; list[j] = temp;
    quicksort(list,left,j-1);
    quicksort(list,j+1,right);
  }
}
```

Parallel Quicksort

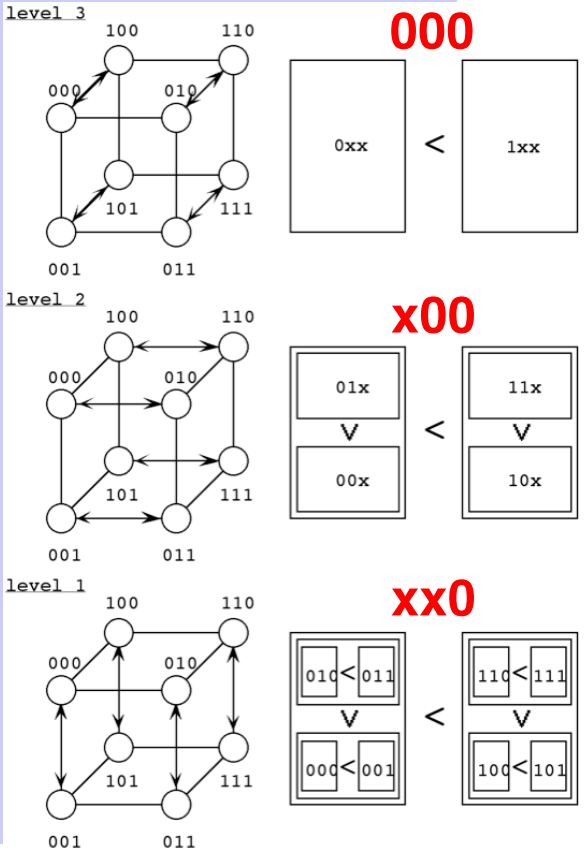
```
bitvalue := 2^(dimension-1);
mask := 2^dimension - 1;
for L := dimension downto 1
begin
  if myid AND mask = 0 then
    choose a pivot value for the L-dimensional subcube;
  broadcast the pivot from the master to the other members of the subcube;
  partition list[0:n_element-1] into two sublists such that
  list[0:j] ≤ pivot < list[j+1:n_element-1];
  partner := myid XOR bitvalue;
  if myid AND bitvalue = 0 then // junior partner
    begin
      send the right sublist list[j+1:n_element-1] to partner;
      receive the left sublist of partner;
      append the received list to my left list
    end
  else // senior partner
    begin
      send (& erase) the left sublist list[0:j] to partner;
      receive the right sublist of partner;
      append the received list to my right list
    end
  n_element := n_element - n_send + n_receive;
  mask = mask XOR bitvalue;
  bitvalue = bitvalue/2;
end
sequential quicksort to list[0:n_element-1]
```

100 → 010 → 001
111 → 011 → 001

# of processes = $2^{dimension}$

pivot = avg(list elements)

Subcube master:



Subcube Broadcast

```
bitvalue = nprocs >> 1;
mask = nprocs - 1;

for (L=dimension; L>=1; L--) {
  ...
  if ((myid & mask) == 0)
    Calculate the pivot as the average of the local list
    element values
  MPI_Bcast(&pivot,1,MPI_INT,0,cube[L][myid/nprocs_cube]);
  ...
  mask = mask ^ bitvalue;    /* Flip the current bit to 0 */
  bitvalue = bitvalue >> 1; /* Next significant bit       */
}
```

MPI_Comm_size(MPI_COMM_WORLD,&nprocs)

right-shift or divide by 2

dimension = log2(nprocs)

bitwise AND

rank within the subcube

bitwise XOR

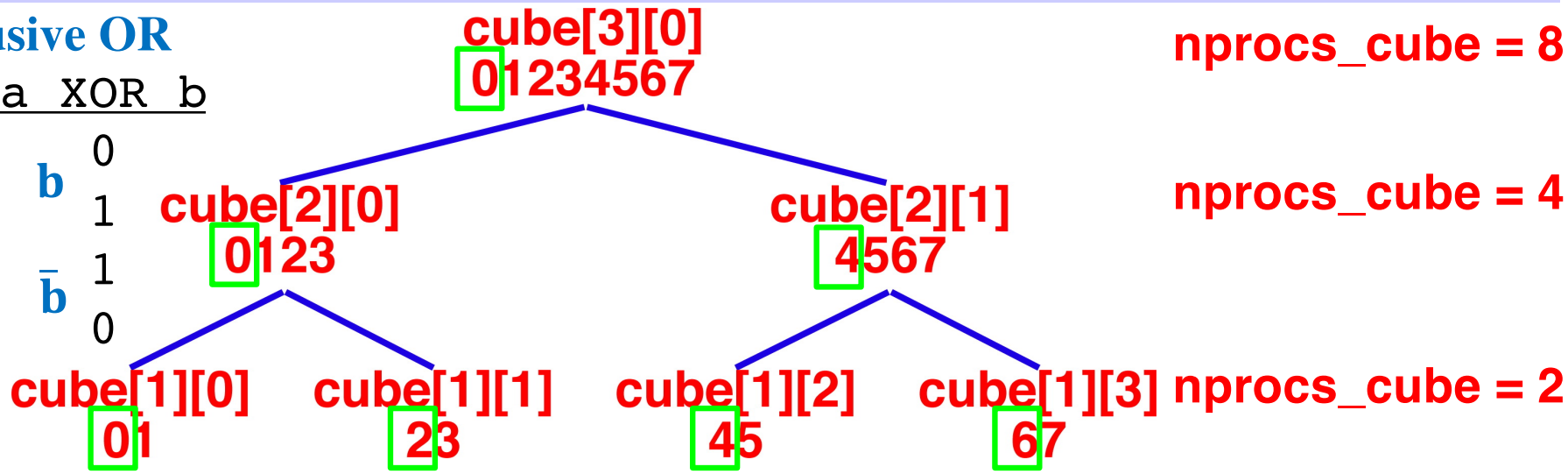
$abcdefg \text{ XOR } 0000100 = abcd\bar{e}fg$

Exclusive OR

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

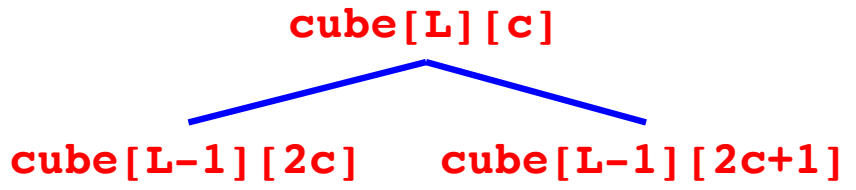
b

$\bar{b}$



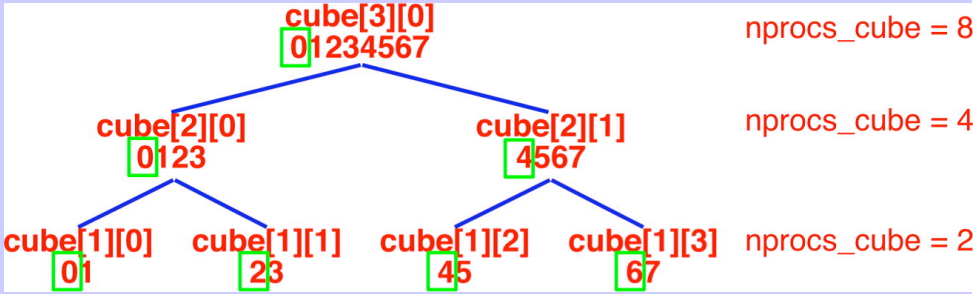
MPI Communicators

- Recursive bisection of processor groups
- Communicator = process group + context
- Rank = 0, 1, ... within each communicator



Max. hypercube dimension (*e.g.*, 5) Max. # of processes (*e.g.*, 32)

```
MPI_Comm cube[MAXD][MAXP];
MPI_Group cube_group[MAXD][MAXP];
...
MPI_Comm_size(MPI_COMM_WORLD,&nprocs_cube);
cube[dimension][0] = MPI_COMM_WORLD;
...
// At each level
MPI_Comm_group(cube[L][c],&(cube_group[L][c]));
nprocs_cube = nprocs_cube/2;
for(p=0; p<nprocs_cube; p++) procs_cube[p] = p;
MPI_Group_incl(cube_group[L][c],nprocs_cube,procs_cube,&(cube_group[L-1][2*c  ]));
MPI_Group_excl(cube_group[L][c],nprocs_cube,procs_cube,&(cube_group[L-1][2*c+1]));
MPI_Comm_create(cube[L][c],cube_group[L-1][2*c  ],&(cube[L-1][2*c  ]));
MPI_Comm_create(cube[L][c],cube_group[L-1][2*c+1],&(cube[L-1][2*c+1]));
MPI_Group_free(&(cube_group[L  ][c    ]));
MPI_Group_free(&(cube_group[L-1][2*c  ]));
MPI_Group_free(&(cube_group[L-1][2*c+1]));
```

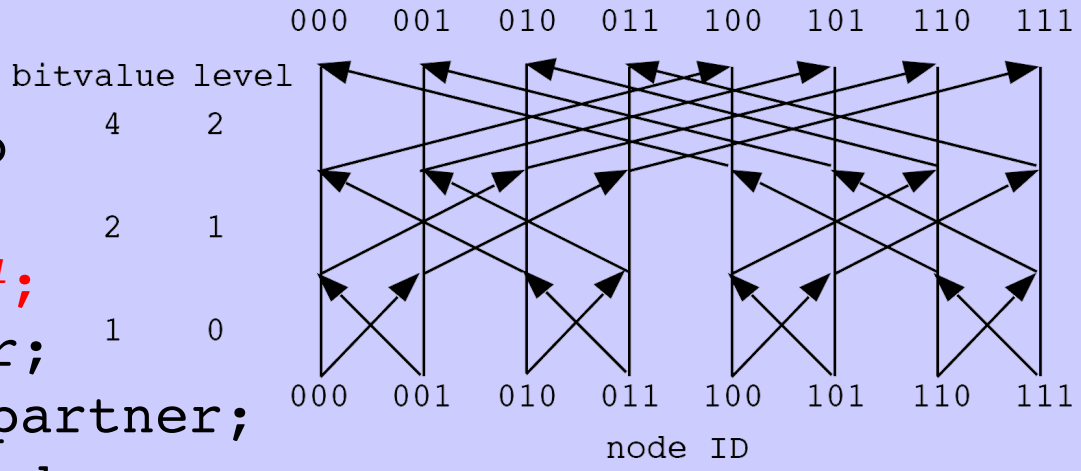


$nprocs\_cube = 2^L$

c = myid/nprocs_cube

Hypercube Template (from MPI Lecture)

```
procedure hypercube(myid, input, log2P, output)
begin
  mydone := input;
  for l := 0 to log2P-1 do
  begin
    partner := myid XOR 2^l;
    send mydone to partner;
    receive hisdone from partner;
    mydone = mydone OP hisdone
  end
  output := mydone
end
```



level l	bitvalue 2^l
0	001
1	010
2	100

Associative operator (*e.g.*, sum, max)

Exclusive OR

a	b	a XOR b	
0	0	0	b
0	1	1	
1	0	1	$\bar{b}$
1	1	0	

abcdefg XOR 0000100 = abcd$\bar{e}$fg

In C, ^ (caret operator) is bitwise XOR applied to int

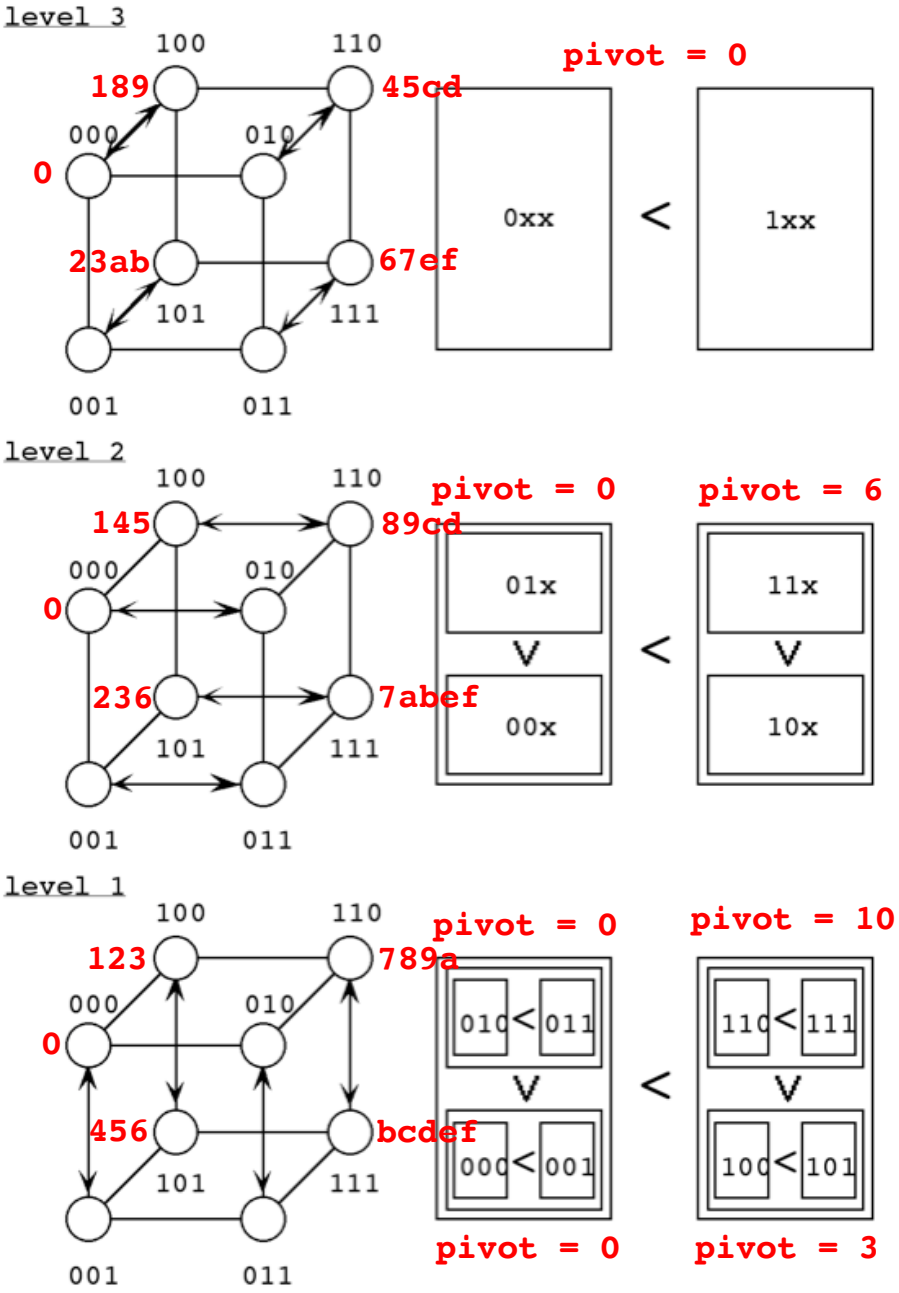
Numerical Result: Random Input

```
Before: Rank  0 : 28  43  72  79
Before: Rank  1 : 96  19  17  71
Before: Rank  2 : 66  67  18  42
Before: Rank  3 : 79  86  73  35
Before: Rank  4 : 88  32  12  72
Before: Rank  5 : 45  61  97  41
Before: Rank  6 : 81  51  41   4
Before: Rank  7 : 94  27  93  44
After:  Rank  0 :  4  12
After:  Rank  1 : 17  18  19  27  28
After:  Rank  2 : 32  35  41  41
After:  Rank  3 : 42  43  44  45  51
After:  Rank  4 : 61  66  67
After:  Rank  5 : 71  72  72  73
After:  Rank  6 : 79  79  81
After:  Rank  7 : 86  88  93  94  96  97
```

inputs:
random list

outputs:
sorted list
more or less balanced

Bad Input: Already Sorted




```
----------------------------------
Before: Rank  0 :  0   1
Before: Rank  1 :  2   3
Before: Rank  2 :  4   5
Before: Rank  3 :  6   7
Before: Rank  4 :  8   9
Before: Rank  5 : 10  11
Before: Rank  6 : 12  13
Before: Rank  7 : 14  15
After:  Rank  0 :  0
After:  Rank  1 :
After:  Rank  2 :
After:  Rank  3 :
After:  Rank  4 :  1   2   3
After:  Rank  5 :  4   5   6
After:  Rank  6 :  7   8   9  10
After:  Rank  7 : 11  12  13  14  15
----------------------------------
```

Hypercube Topology

- **Hamming distance:** The total number of bit positions at which two binary numbers differ.
- In a hypercube network topology, two nodes are connected if their Hamming distance is 1. The *connectivity* of a d-dimensional (or n-node) hypercube is thus d. Since each link can change only one digit, the *diameter* is d or $\log_2 n$.

Recursive hypercube construction algorithm

(1) A one-dim. hypercube has two connected nodes 0 & 1.

(2) A (d+1)-dim. hypercube is defined from a d-dim. hypercube:

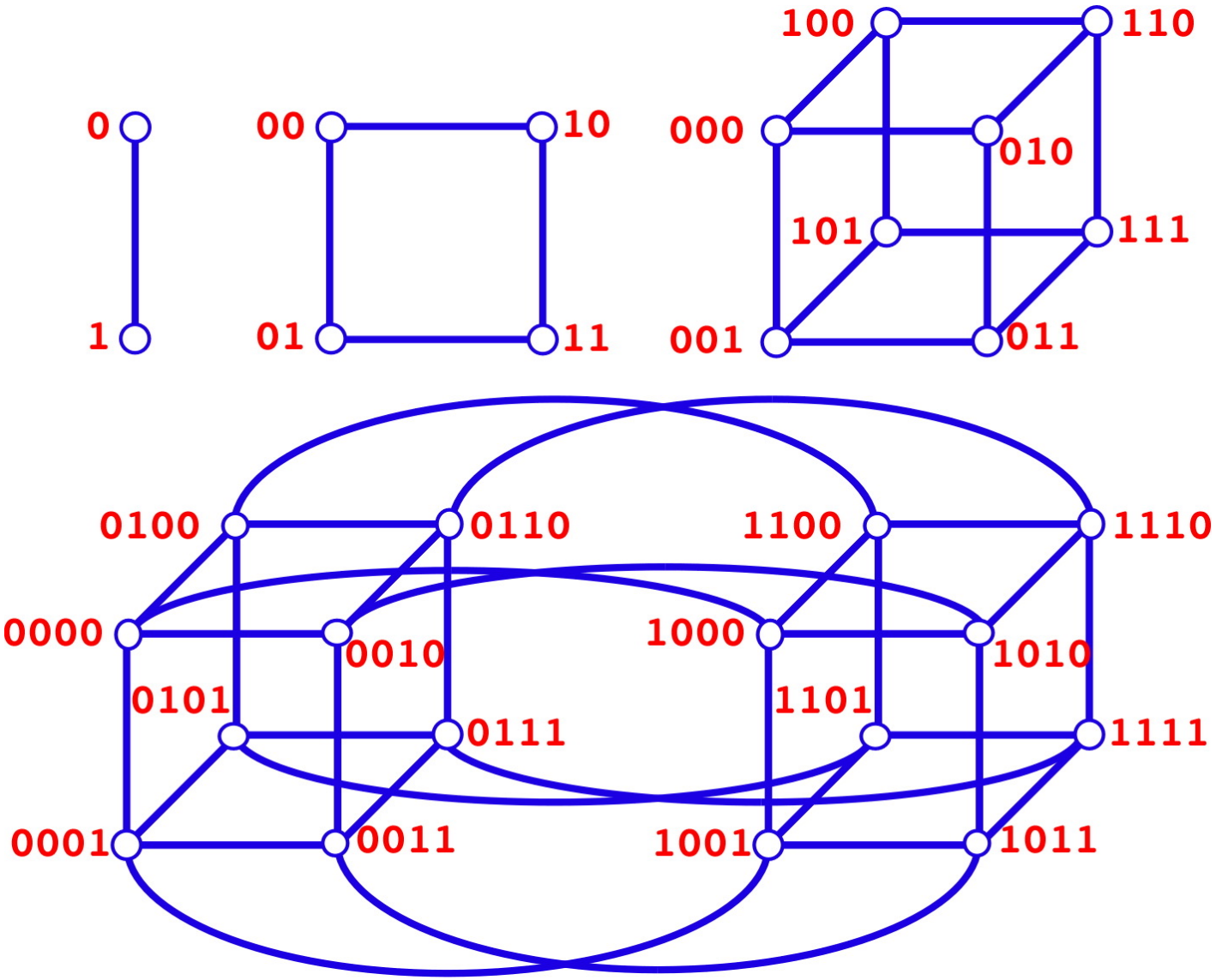
a. Duplicate the d-dim. hypercube including node numbers.

b. Create links between nodes with the same number in the original & duplicate.

c. Append a binary 1 to the left of each node number in the duplicate, & a binary 0 to left of each node number in the original.

See Grama'03, Chap. 2

Hypercube



- $\log_2 n$ network interfaces required per node for an n-node parallel computer
- Hypercube algorithms only use direct network links on a hypercube

Analysis of Parallel Quicksort

```
for L := dimension downto 1                                   log p
begin
  if master then
    choose a pivot value for the L-dimensional subcube;       n/p
  broadcast the pivot from the master to the subcube members;  [1,log p]
  partition list[0:n_element-1] into two sublists such that   n/p
  list[0:j] ≤ pivot < list[j+1:n_element-1];
  if lower_partner then
    begin
      send the right sublist list[j+1:n_element-1] to partner;
      receive the left sublist of partner;                    n/p
    end
  else if higher_partner then
    begin
      send the left sublist list[0:j] to partner;
      receive the right sublist of partner;                   n/p
    end
  n_element := n_element - n_send + n_receive;
end
sequential quicksort to list[0:n_element-1]                   (n/p)log(n/p)
```

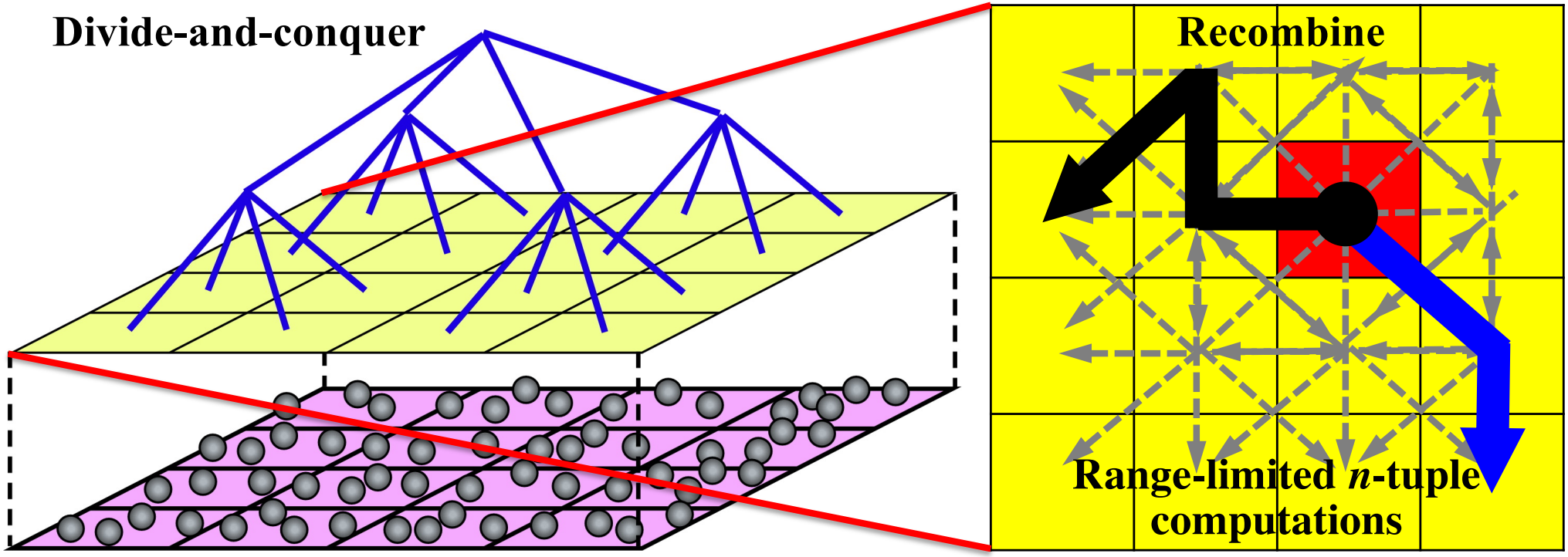
$$T_{\text{average}} = O\left(\frac{n}{p}\log\frac{n}{p}\right) + O\left(\frac{n}{p}\log p\right) + O\left(\log^2 p\right)$$

$$\sum_{l=1}^{\log P} l = \frac{\log P(\log P + 1)}{2} = O(\log^2 P)$$

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―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（**recombine**）することができ、そこに創造（**creativity**）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）

Divide-Conquer-Recombine Algorithms



M. Kunaseth *et al.*, *ACM/IEEE SC13* ('13)

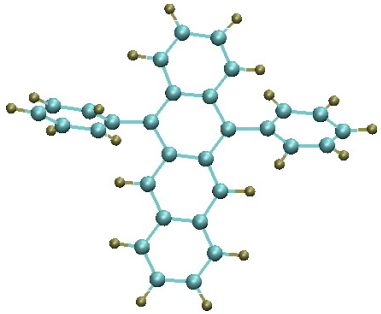
Globally informed local DC-DFT solutions are used in the recombine phase as compact bases to synthesize global properties in broad applications:

- **High-order inter-molecular-fragment correlation** [S. Tanaka *et al.*, '13]
- **Global frontier orbitals (HOMO & LUMO)** [S. Tsuneyuki *et al.*, '09, '13]
- **Global charge migration** [H. Kitoh-Nishioka *et al.*, '12; C. Gollub *et al.*, '12]
- **Global exciton dynamics** [W. Mou *et al.*, '13]

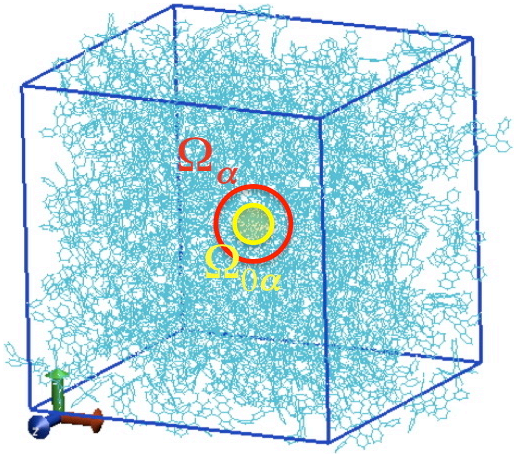
F. Shimojo *et al.*, *J. Chem. Phys.* **140**, 18A529 ('14)

Simulating SF in Amorphous DPT

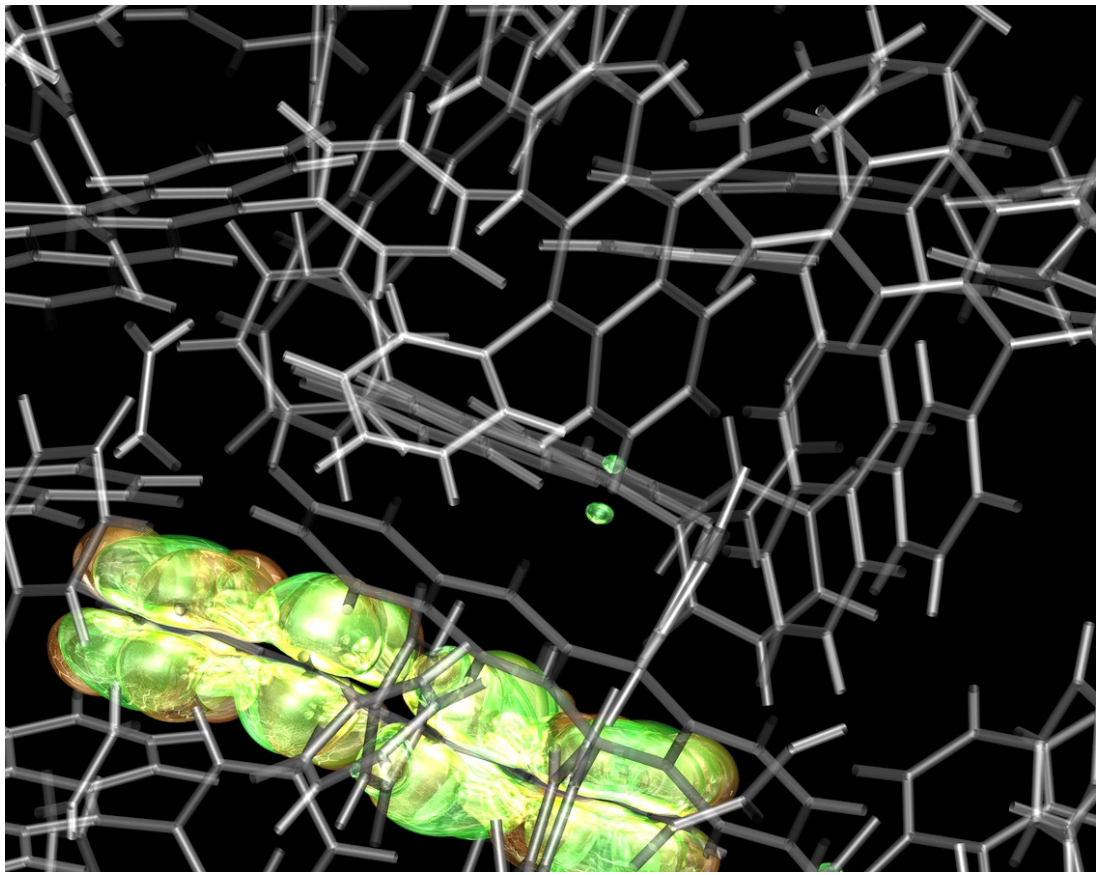
- Move up from molecules to microstructures
- Challenge: Unprecedented 10^4-atom NAQMD simulation
- Computational approach: Divide-conquer-recombine (DCR) NAQMD



DPT molecule



Amorphous DPT



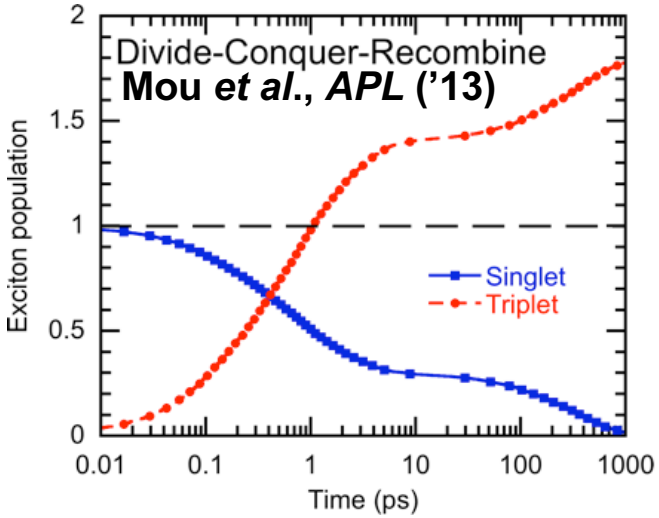
Quasi-electron

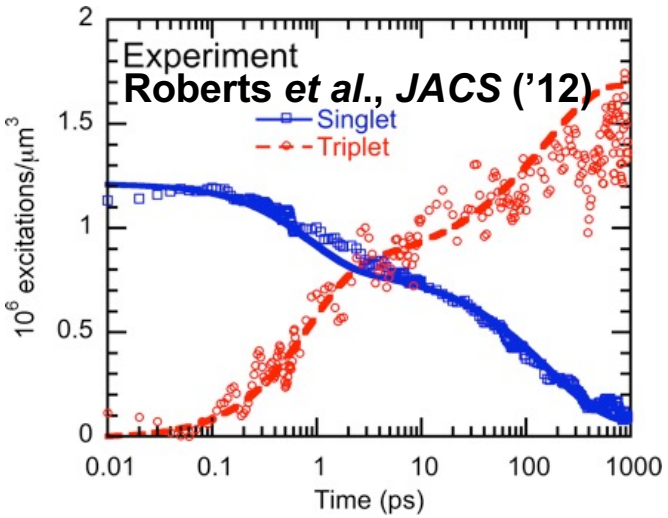
Quasi-hole

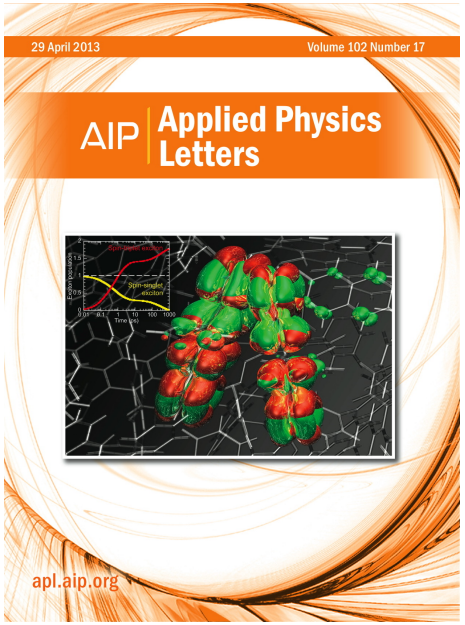
- Divide-conquer-recombine NAQMD (phonon-assisted exciton dynamics) + time-dependent perturbation theory (singlet-fission rate) + kinetic Monte Carlo calculations of exciton population dynamics in 6,400-atom amorphous DPT

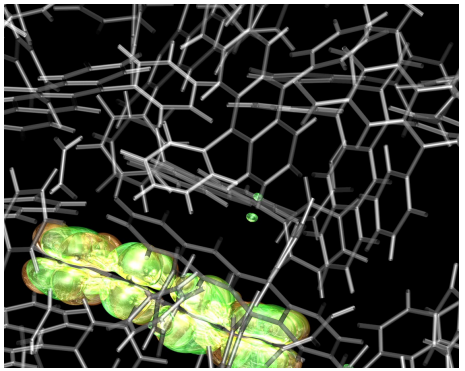
Singlet-Fission Hot Spot

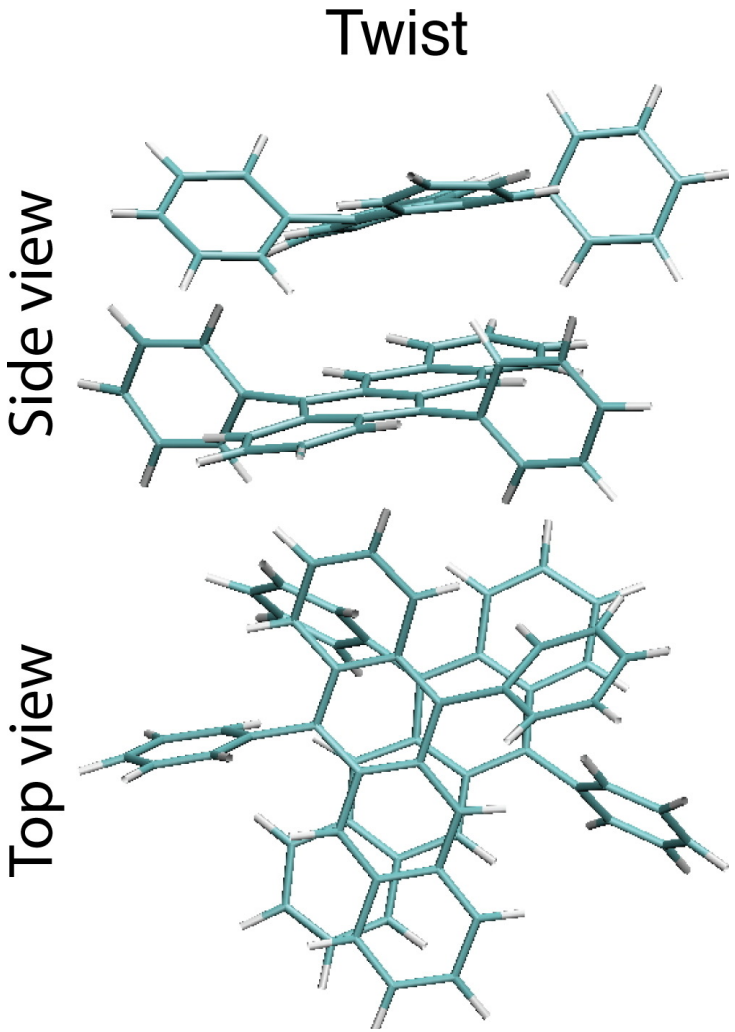
- **Quantum molecular dynamics simulations not only reproduced experimentally measured exciton population dynamics but also revealed unknown molecular geometry of singlet fission hot spots**











What We Have Learned Here

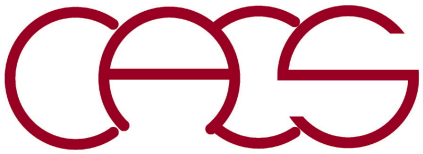
- **Divide-conquer(-recombine) will continue to scale on future massively parallel computing platforms.**
- **Divide-&-conquer (DC) algorithm recursively subdivides the problem into sub-problems (*e.g.*, recursive bisection in quicksort; quadtree & octree, respectively, in 2D & 3D fast multipole method).**
- **DC algorithm often uses collective operations (*e.g.*, broadcast & reduction) within sub-problem.**
- **Collective sub-problem operations are best implemented using MPI Communicator construct.**

Divide-&-Conquer and Systems

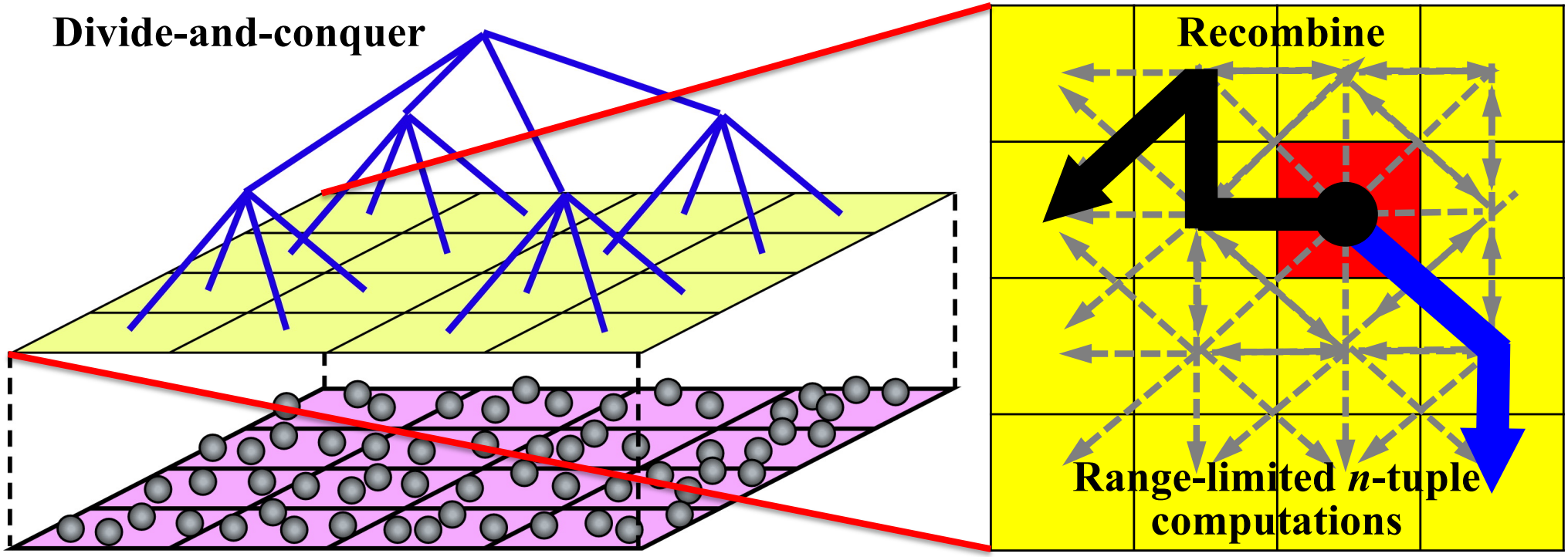
Aiichiro Nakano

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



Metascalable Divide-Conquer-Recombine

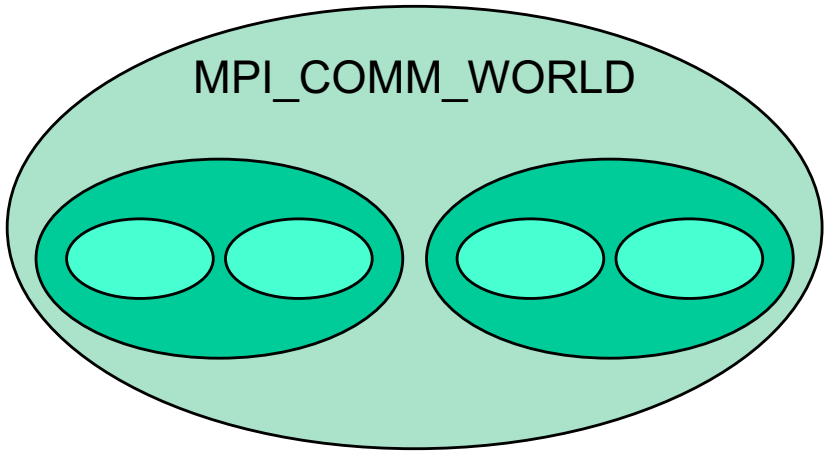


M. Kunaseth *et al.*, *ACM/IEEE SC13* ('13)

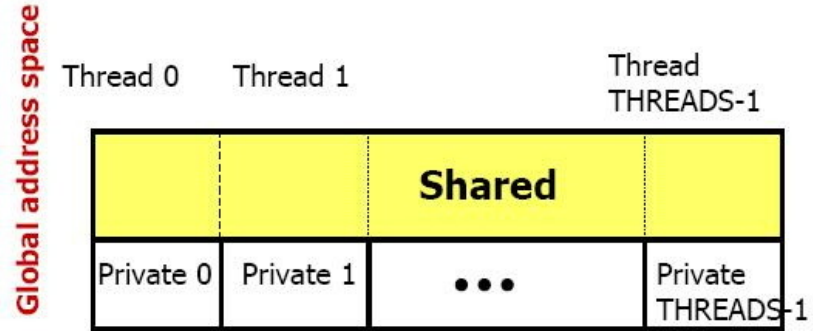
- **Globally informed local DC-DFT solutions are used in the recombine phase as compact bases to synthesize global properties in broad applications**
- **Continue to scale on future computers (*i.e.*, metascalable) with minimal architectural assumptions (*i.e.*, support for tree communication topology)**

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

System Support for D-&-C



Recursive Communicator partition in MPI



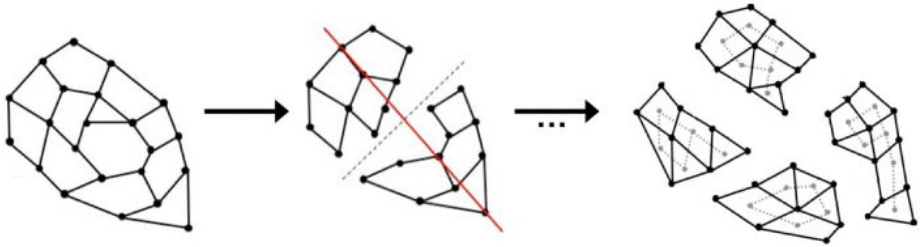
UPC: Distributed Shared Memory Programming, T. El-Ghazawi *et al*. (Wiley, '03)

Nested private variables in partitioned global address space (PGAS) languages?

```
shared a;
...
parallel {
  private b;
  ...
  parallel {
      private c;
      ...
    }
  }
}
```

Tarek A. El-Ghazawi
The George Washington University
Hierarchical Locality and Parallel Programming in the Extreme Scale Era

Implementation mechanism: recursive graph partition



Domain-specific D-&-C language? *cf*. MapReduce

Sample Application: $O(N)$ Deep Learning

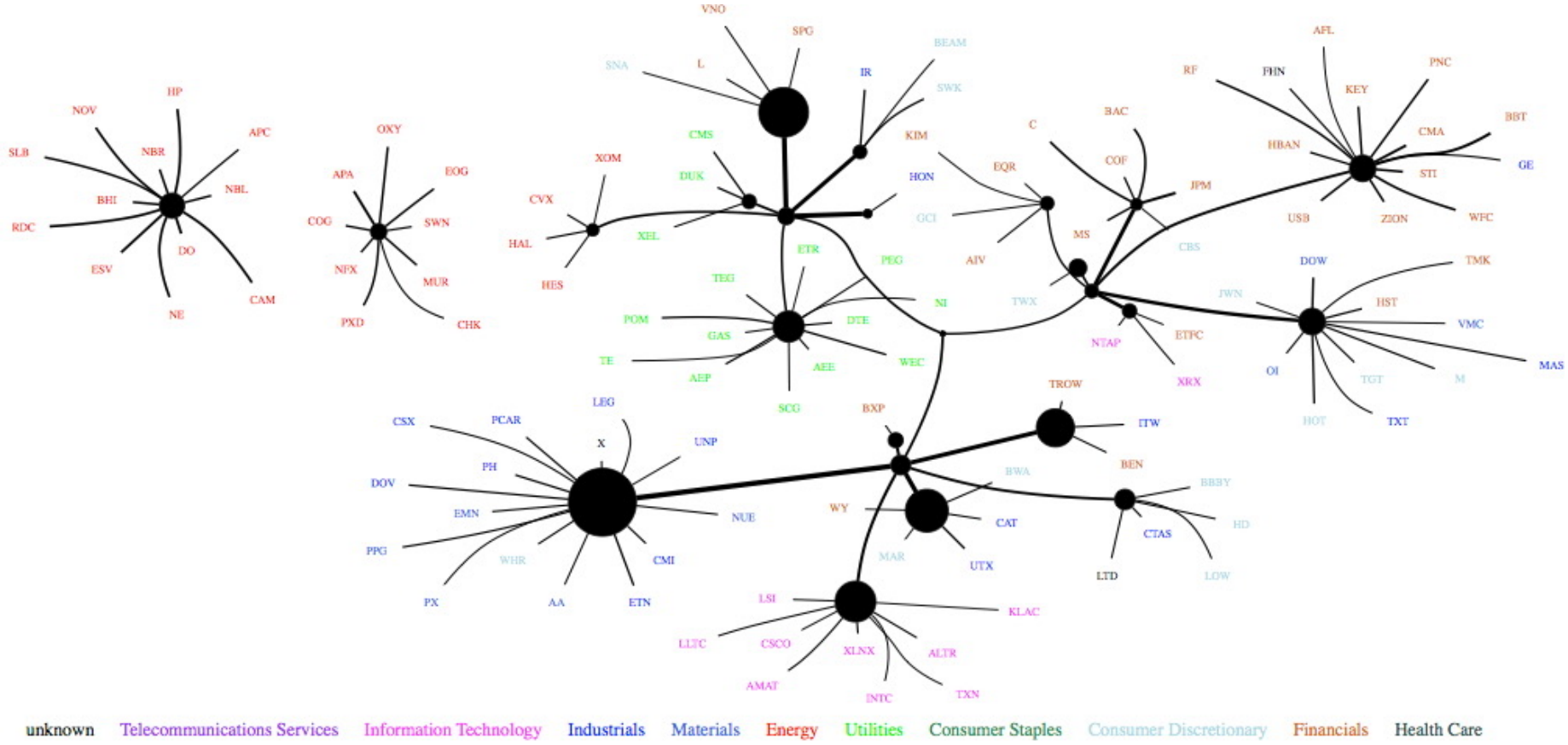


Figure 4: A thresholded graph showing the overall structure of the representation learned from monthly returns of S&P 500 companies. Stock tickers are colored (online) according to their GICS sector. Edge thickness is proportional to mutual information and node size represents multivariate mutual information among children.

G. Ver Steeg & A. Galstyan, *AISTATS 2015* ('15)

Graph Embedding in Semantic Space

Synthesized Classifiers for Zero-Shot Learning

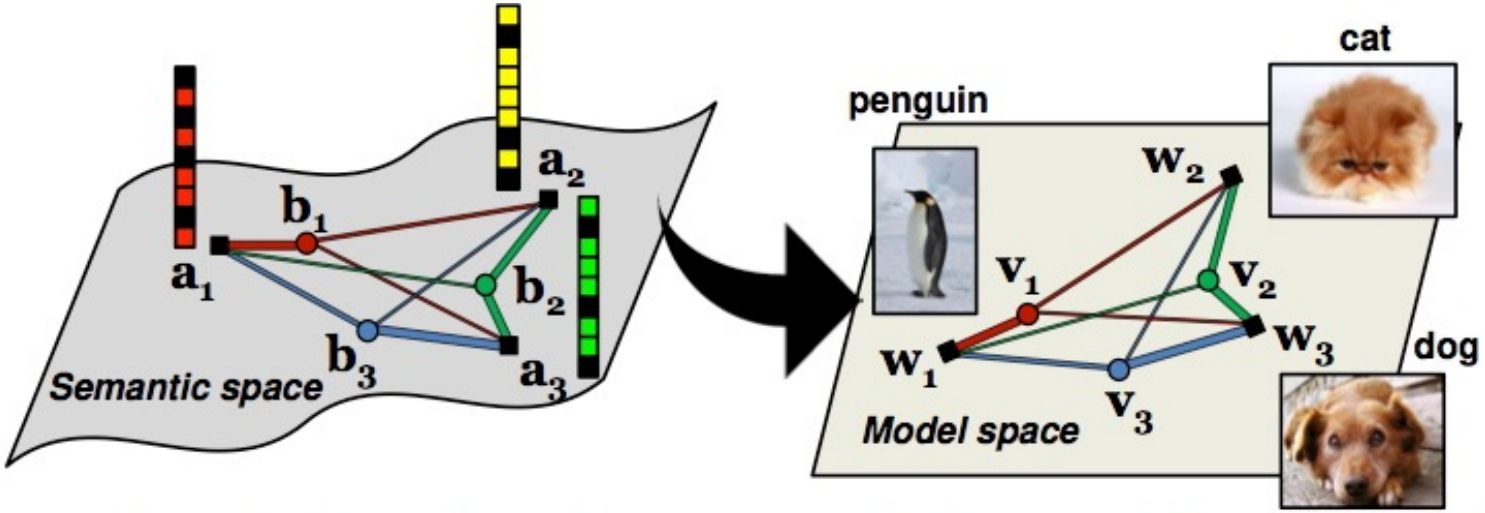


Figure 1: Illustration of our method for zero-shot learning. Object classes live in two spaces. They are characterized in the semantic space with semantic embeddings ($\boldsymbol{a}$s) such as attributes and word vectors of their names. They are also represented as models for visual recognition ($\boldsymbol{w}$s) in the model space. In both spaces, those classes form weighted graphs. The main idea behind our approach is that these two spaces should be aligned. In particular, the coordinates in the model space should be the projection of the graph vertices from the semantic space to the model space — preserving class relatedness encoded in the graph. We introduce adaptable phantom classes ($\boldsymbol{b}$ and $\boldsymbol{v}$) to connect seen and unseen classes — classifiers for the phantom classes are bases for synthesizing classifiers for real classes. In particular, the synthesis takes the form of convex combination.

- **Zero-shot learning = recognize an object in an unseen class**
- **Recursive graph embedding?**

S. Changpinyo *et al.*, *CVPR 2016* (IEEE, '16)