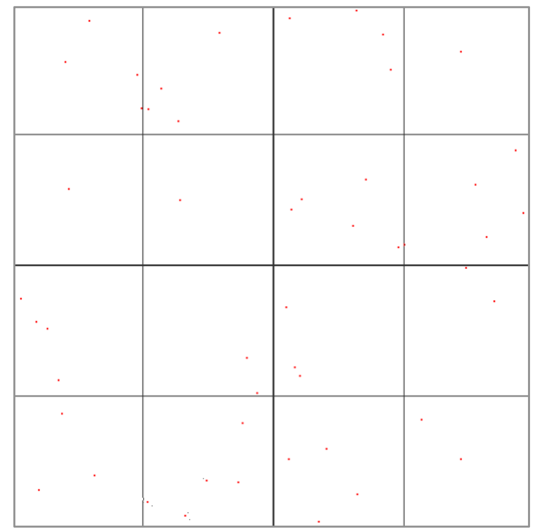
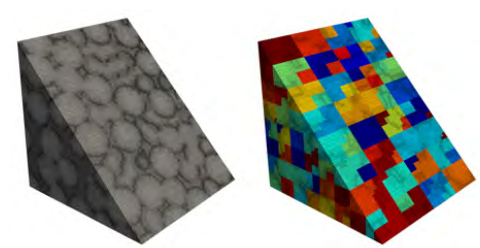
Algorithms and architectures for N-body methods

George Biros

XPS-DSD CCF-1337393

A2MA - Algorithms and Architectures for Multiresolution Applications 2014-2017





Outline

- Significance
- Sketch or algorithm
- Kernels
- Challenges
- Focus on HPC & single-node performance

INSTITUTE FOR COMPUTATIONAL ENGINEERING & SCIENCES
THE UNIVERSITY OF TEXAS AT AUSTIN

Team

PIs:

George Biros (ICES)

Robert van de Geijn (CS)

Andreas Gerstlauer (ECE)

Lizy John (ECE)

Students:

Mochamad Asri, Jianyu Huang, Ahmed Khawaja,

Dhairya Malhotra, Jiajun Wang, Chenhan Yu

Electrical and Computer Engineering

THE UNIVERSITY OF TEXAS AT AUSTIN COMPUTERSCIENCE

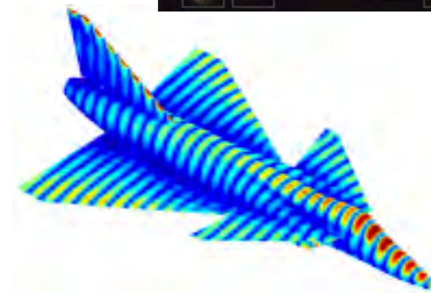
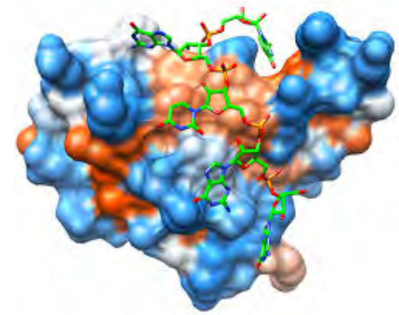
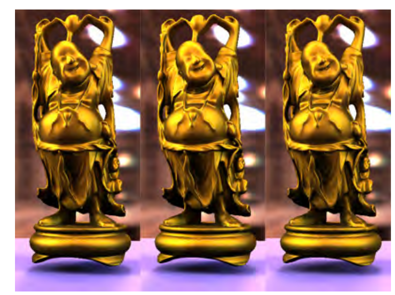
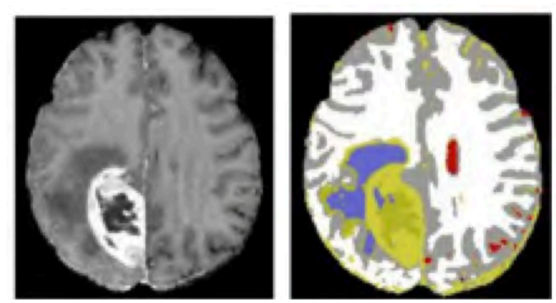
ICES

Scalability and performance of tree algorithms

- N-body: treecodes, Fast Multipole Methods
- Finite elements
- Goal: flop/watt efficiency
- Performance
  - Characterization / Optimization
  - Redesign of concurrent/work-optimal algorithms
- Integration with special-purpose ASIC
  - Linear Algebra Processor (LAP)
  - Novel hardware and software primitives

Applications of N-body algorithms

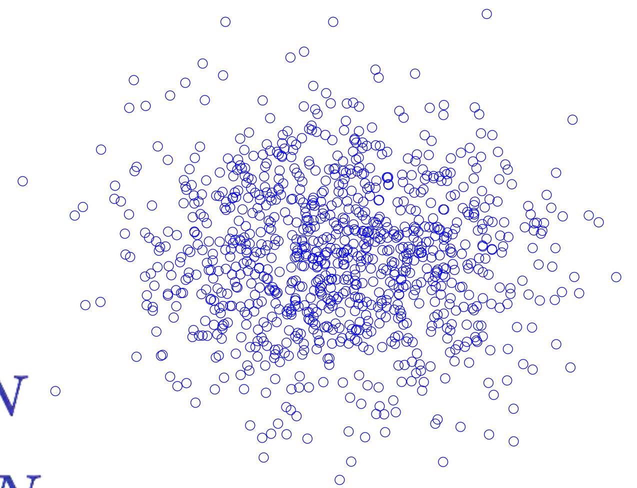
- Cosmology – gravity
- Electrostatics – semiconductors
- Scattering – remote sensing
- Biology – protein interactions
- Mechanics – complex/fluids
- Graphics – radial basis functions
- Geostatistics – kringing
- Signal analysis – non-uniform FFT
- UQ – Gaussian processes
- Machine learning – kernel methods

Sketch of N-body algorithms

Essentially a matrix – vector multiplication problem

N-body methods, Kernel sums

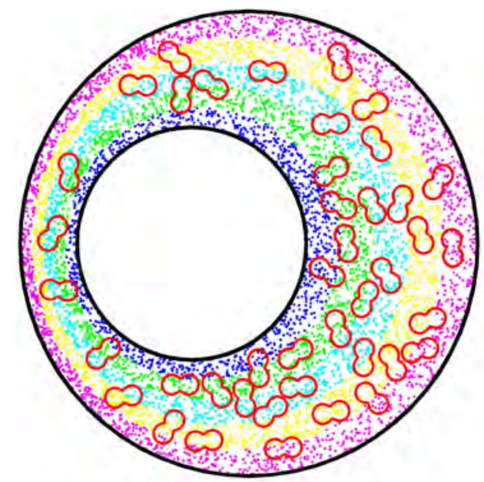


Input

N points in $\mathbb{R}^d$: $\quad x_1, \ldots, x_N$

N densities in $\mathbb{R}$: $\quad w_1, \ldots, w_N$

Output



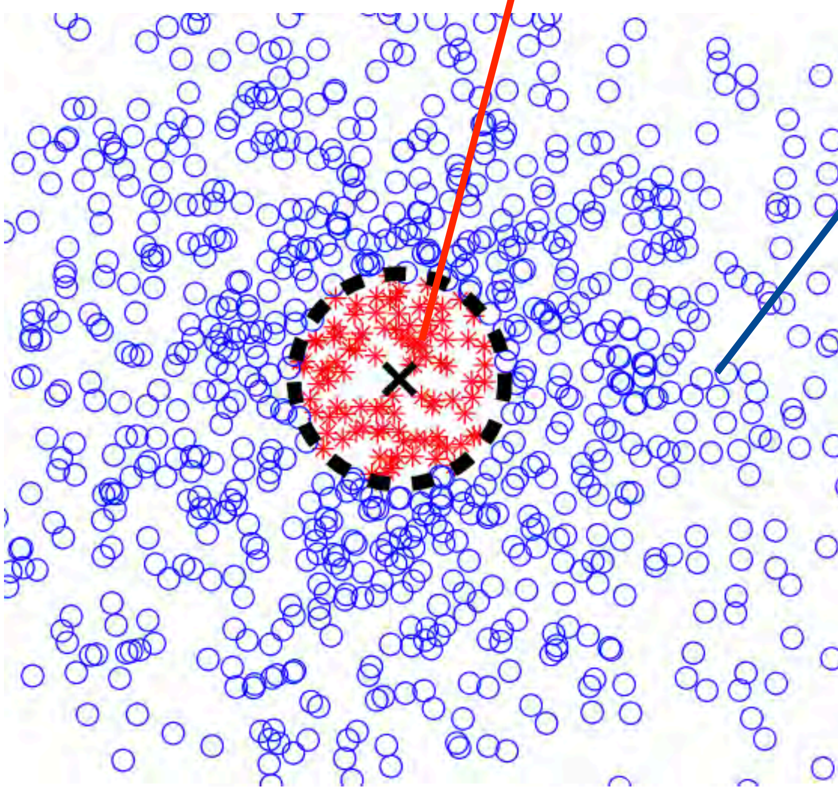
N potentials in $\mathbb{R}$: $\quad u_1, \ldots, u_N$

$$u_i = \sum_{j=1}^{N} G(x_i, x_j) w_j$$

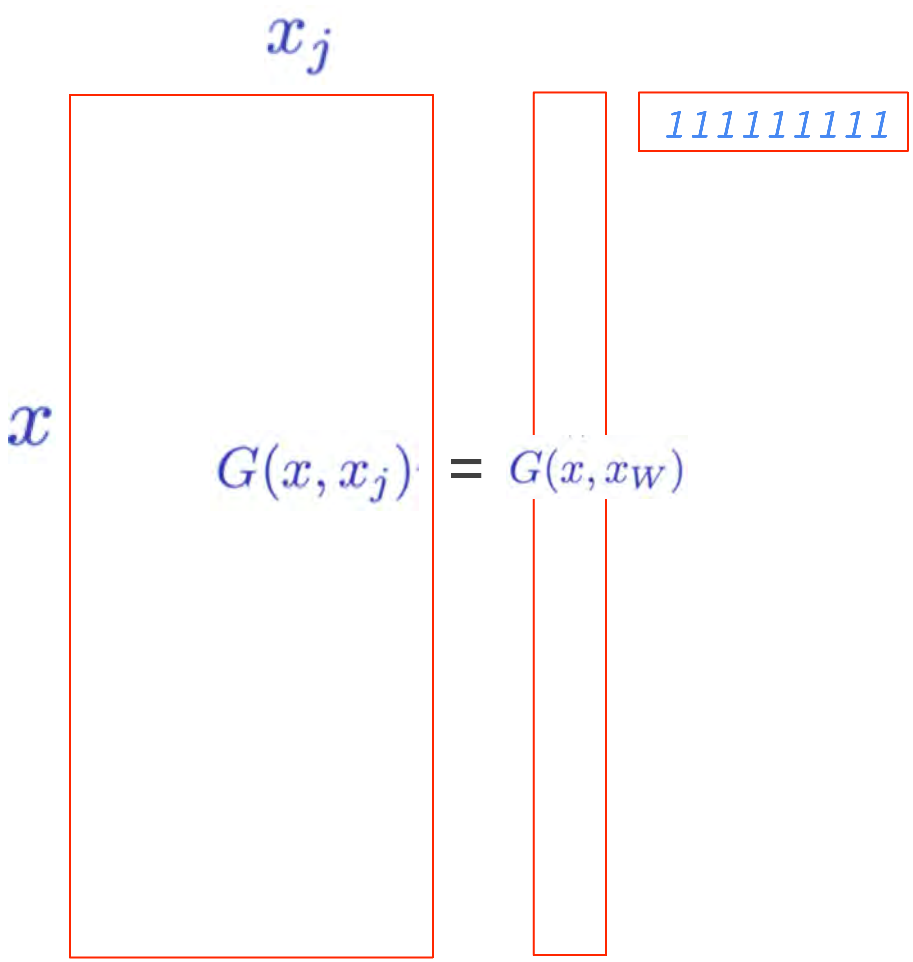
$$G(x_i, x_j) = \exp\left(-\frac{1}{2} \frac{\|x_i - x_j\|_2^2}{h^2}\right)$$

Near-far field decomposition

$$u_i = \sum_{\substack{j=1 \\ j \neq i}}^{N} G(x_i, x_j) w_j = \sum_{j \in \mathrm{near}(i)} G_{ij} w_j + \sum_{j \in \mathrm{far}(i)} G_{ij} w_j$$



Key idea in N-body: low-rank far field approx

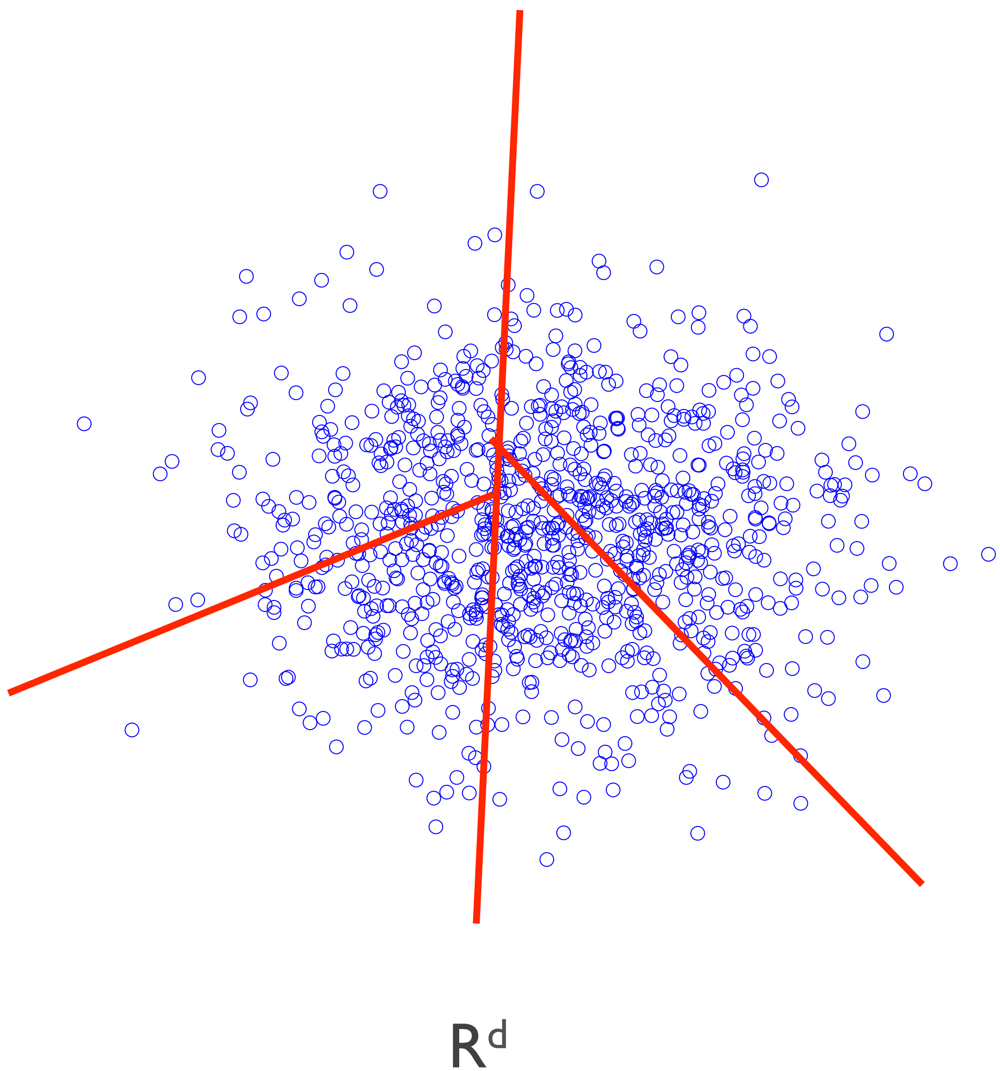


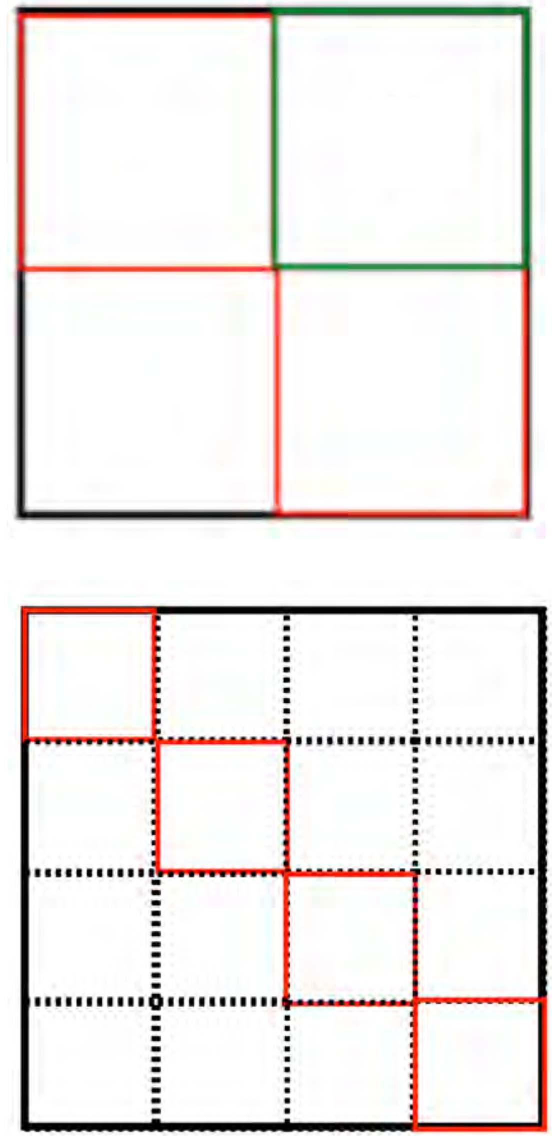
x: Target point
x_j: source location
w_j: weight for sources

$$u(x) = \sum_j G(x, x_j) w_j$$

1. compute $W = \sum_i w_j$
2. compute $x_W = \frac{\sum_j x_j w_j}{W}$
3. $u(x) \approx G(x, x_W) W$

Matrix partitioning

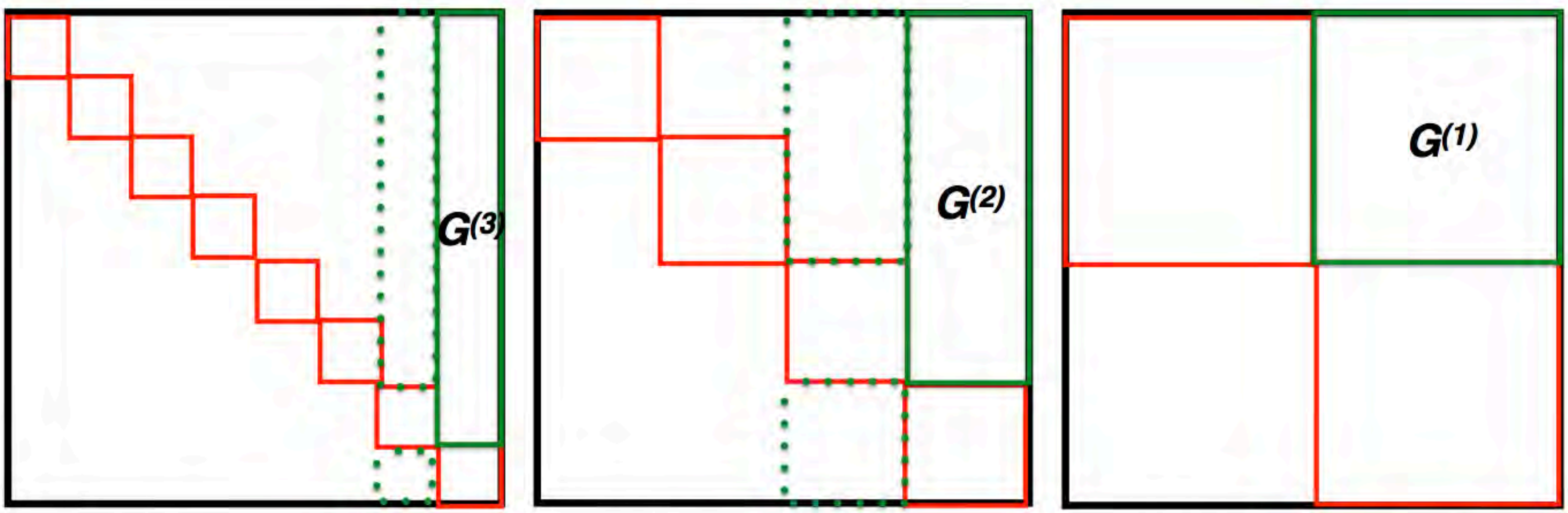




Matrix partitioning

Algebraic view

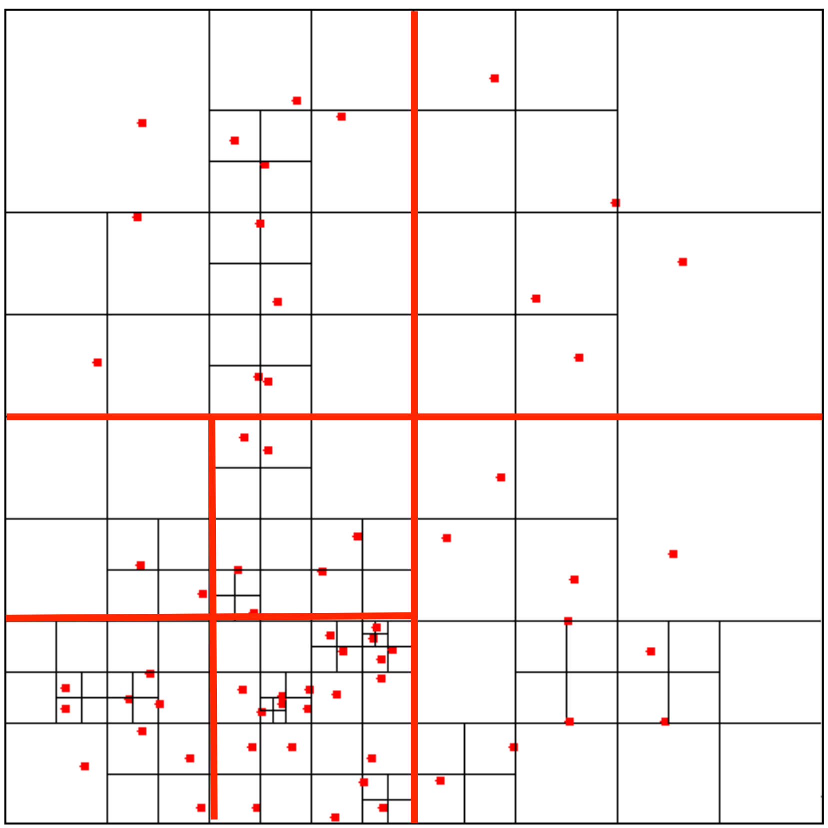
$$u_i = \sum_{j \in \text{diagonal}(i)} G_{ij} w_j + \sum_{j \in \text{off-diagonal}(i)} G_{ij} w_j$$



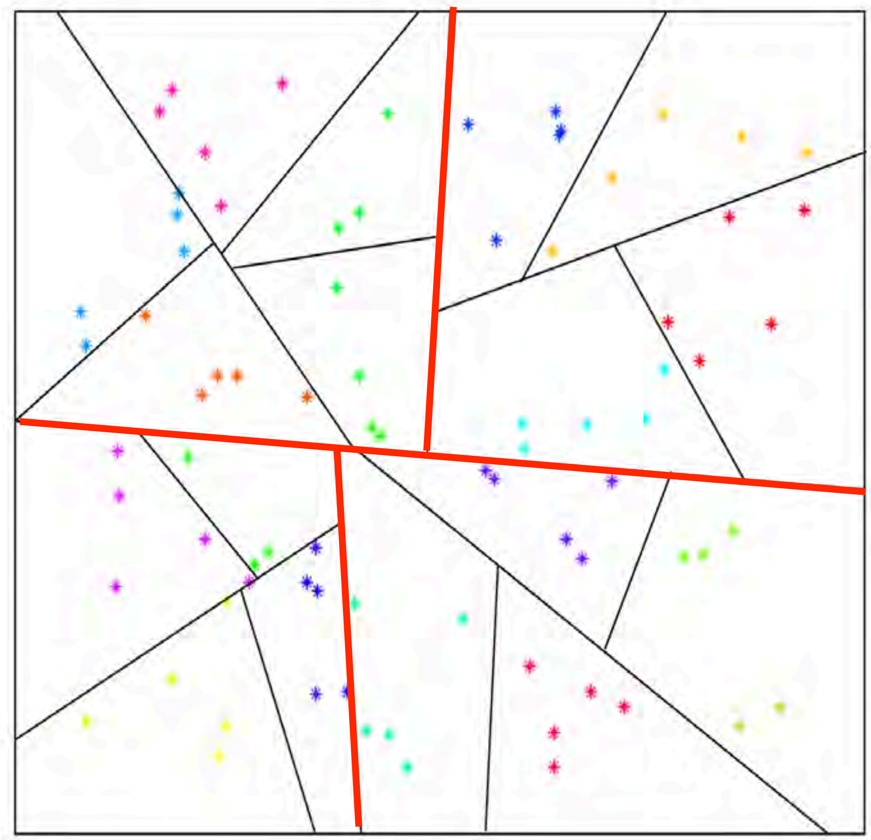
(a) **Level 3.** (b) **Level 2.** (c) **Level 1.**

Regular decompositions

Low dimensions



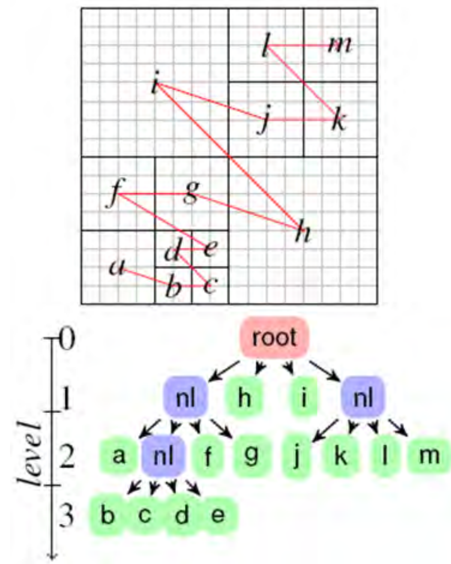
High dimensions

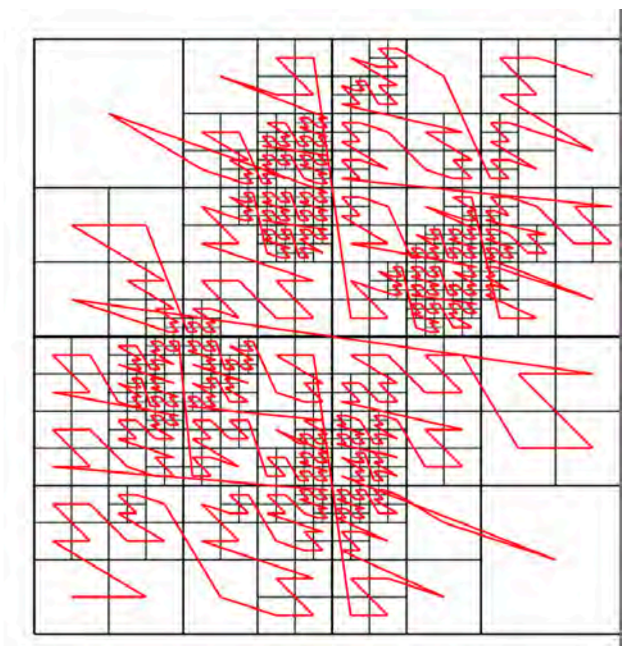


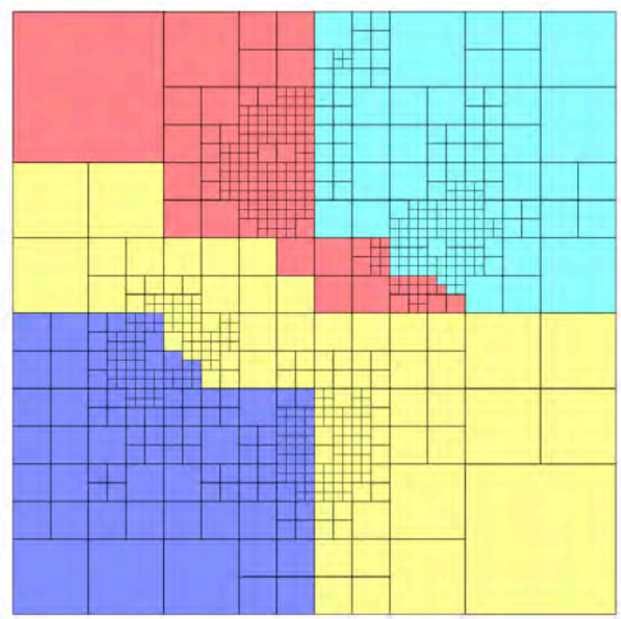
Tree construction concurrency in low dimensions

Bottom-up construction of tree

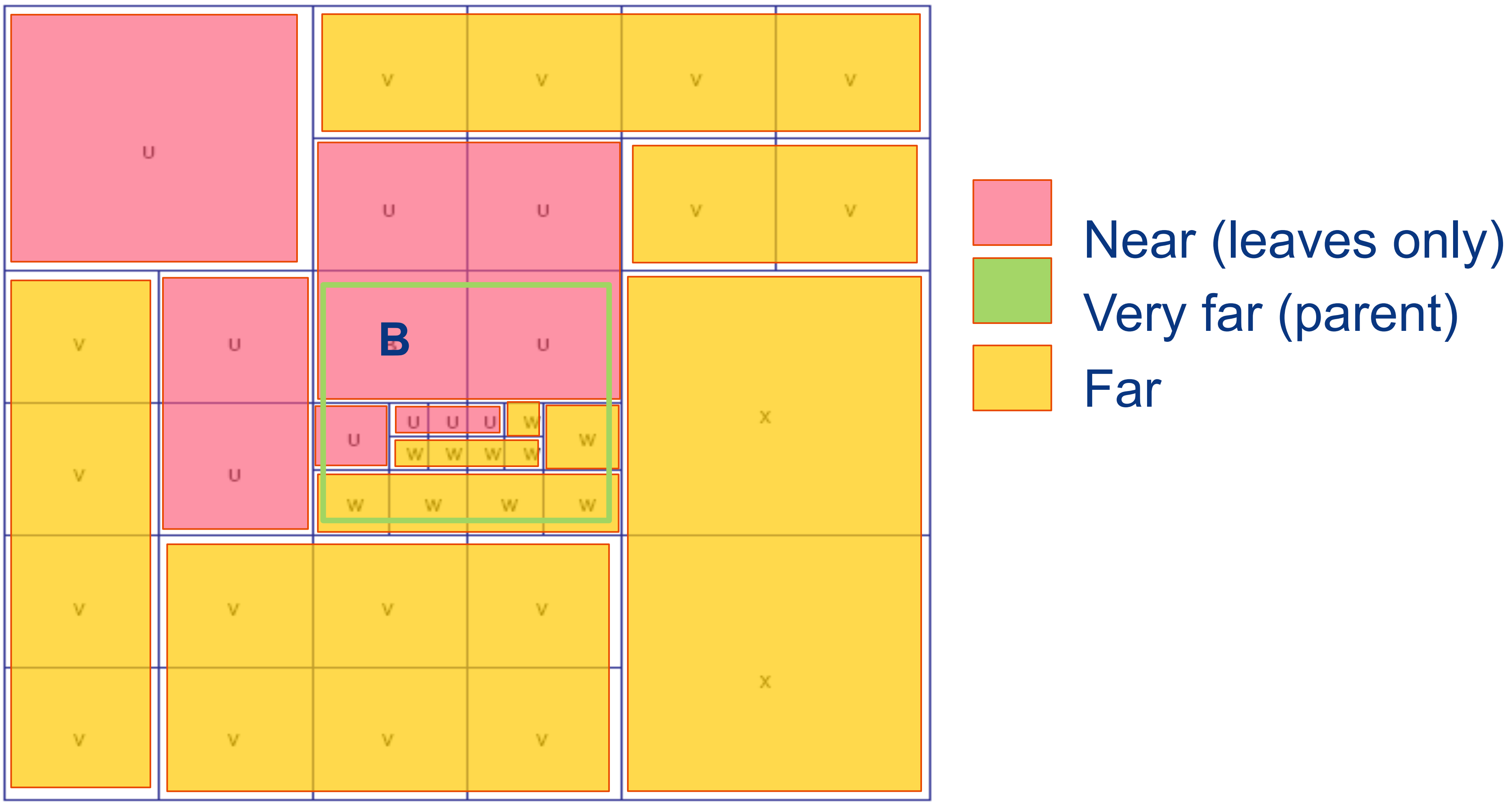
- Space-filling curves for partitioning; sorting; merging; scanning



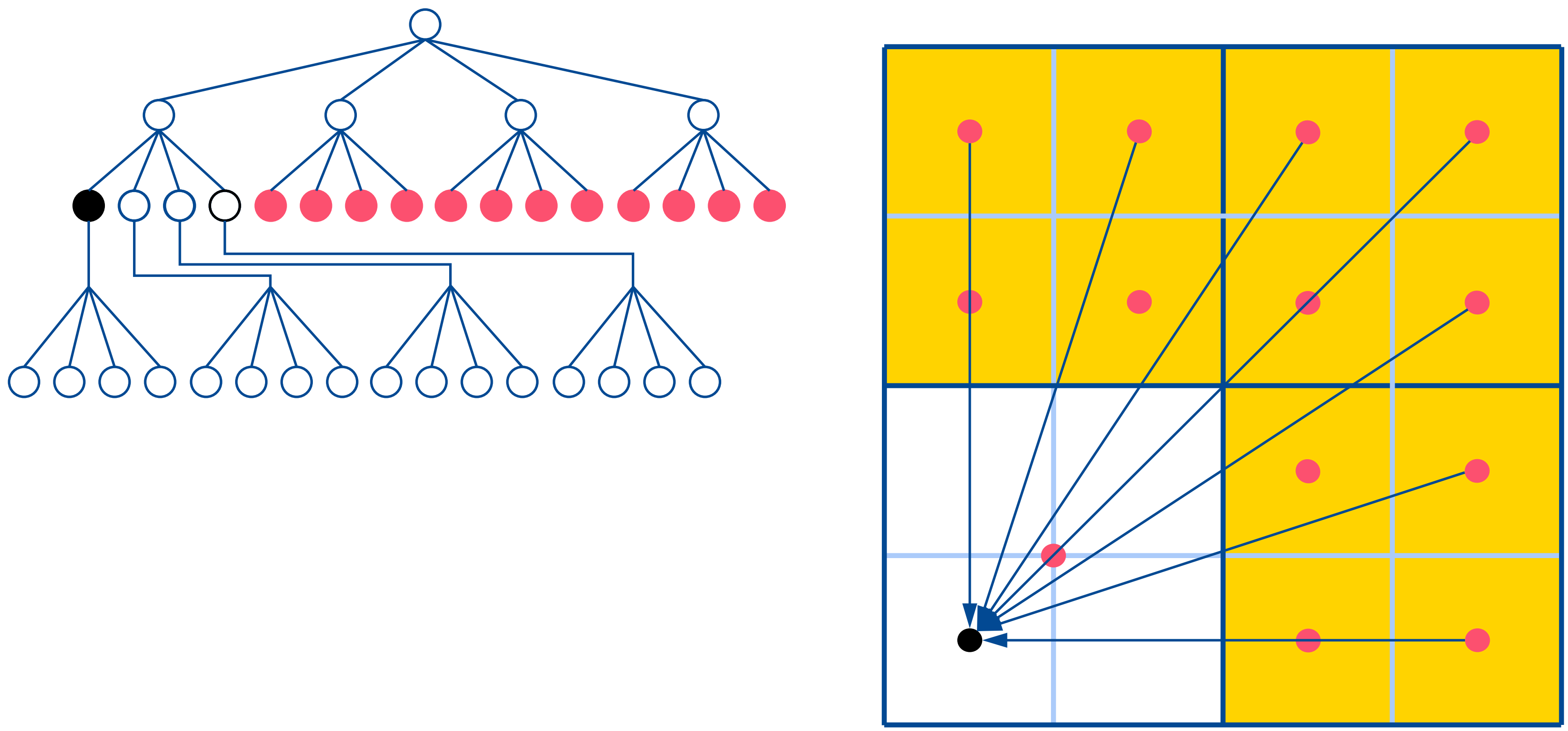




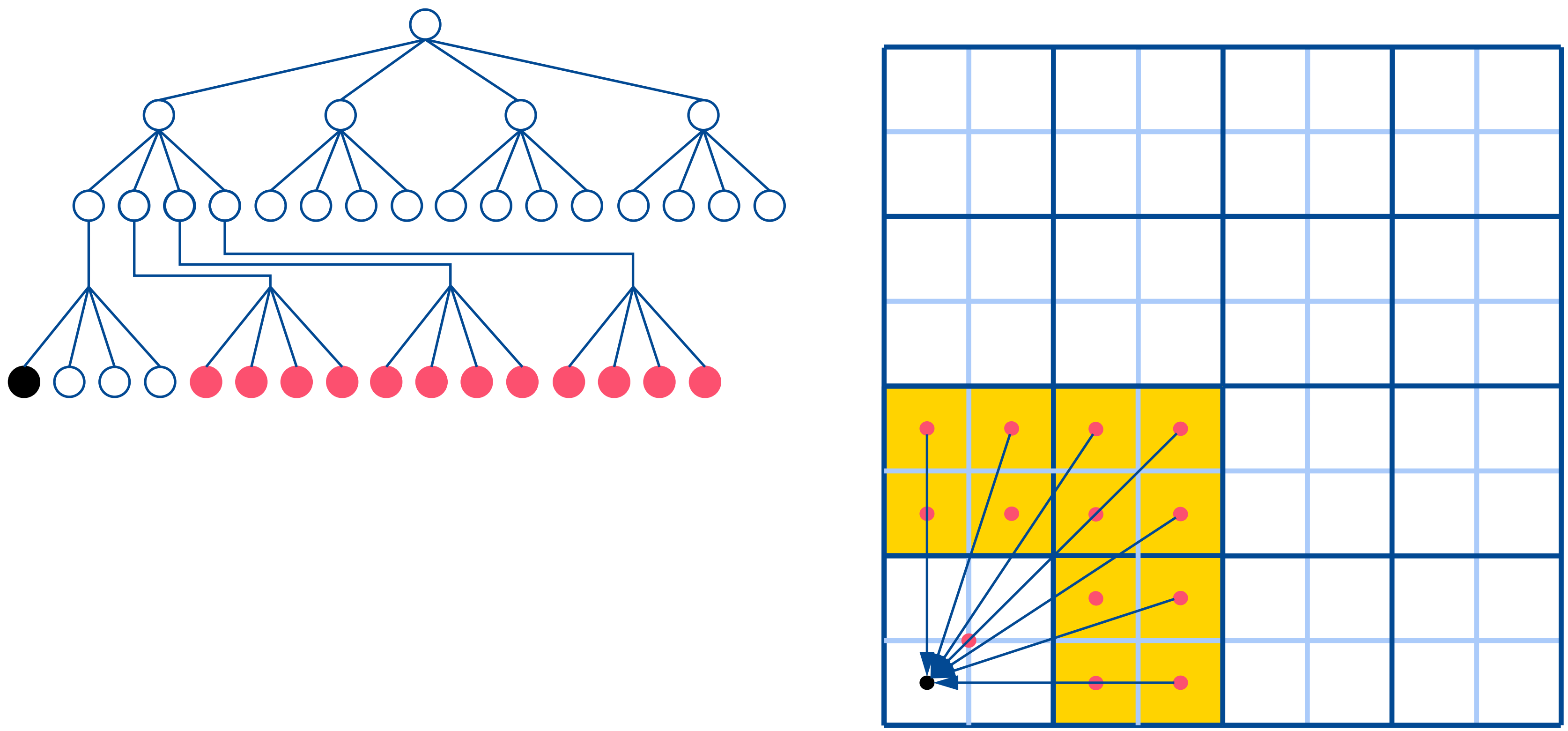
For each node: construction of neighborhoods



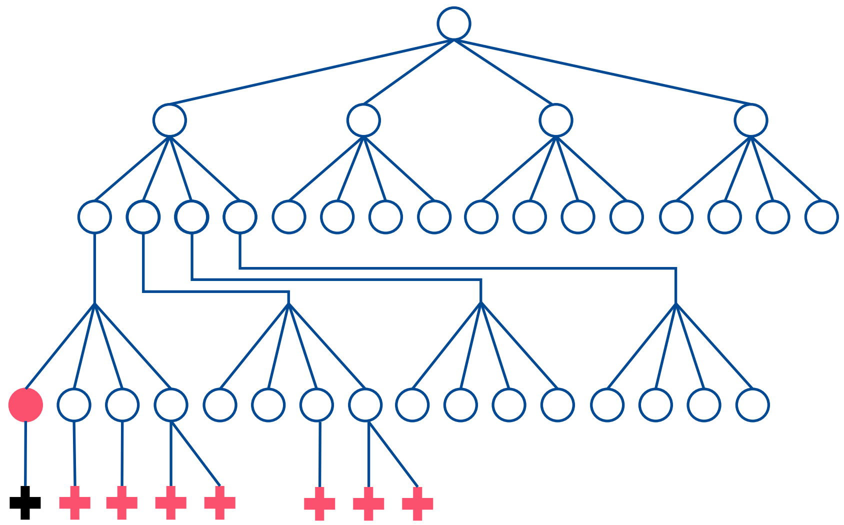
Far-field evaluation

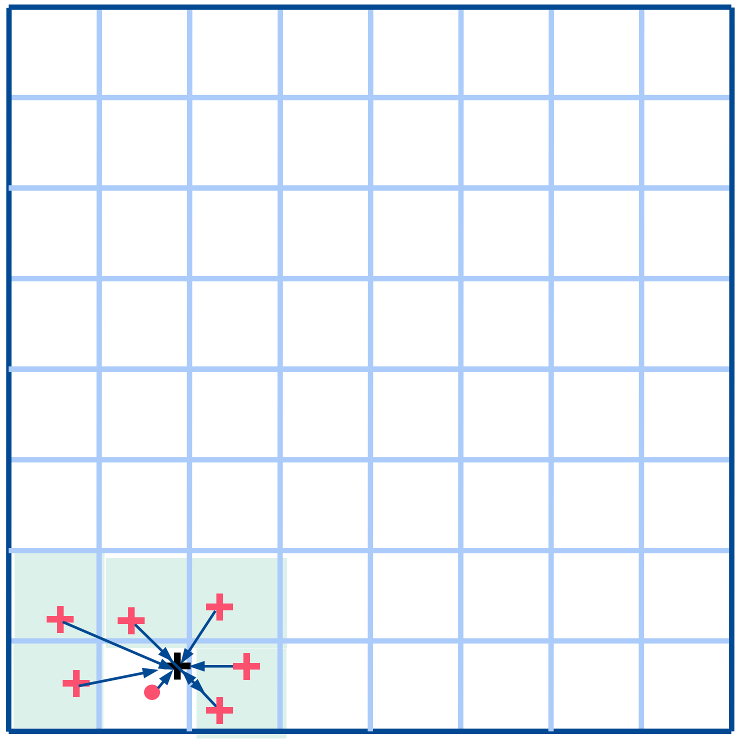


Far field evaluation

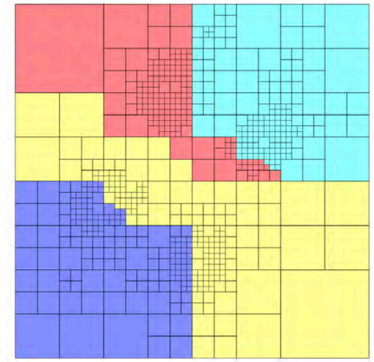


Near-field evaluation

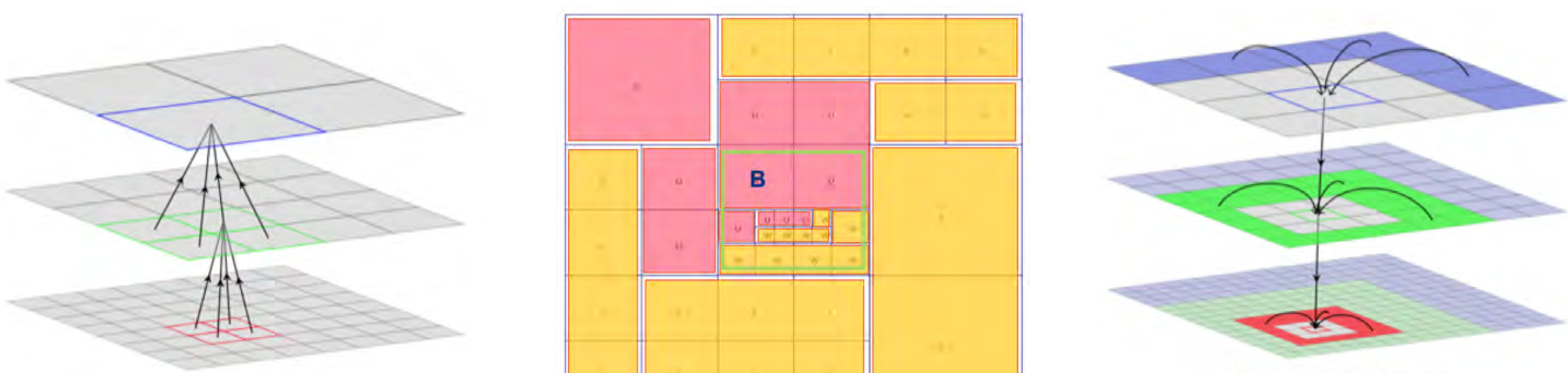




Overall algorithm

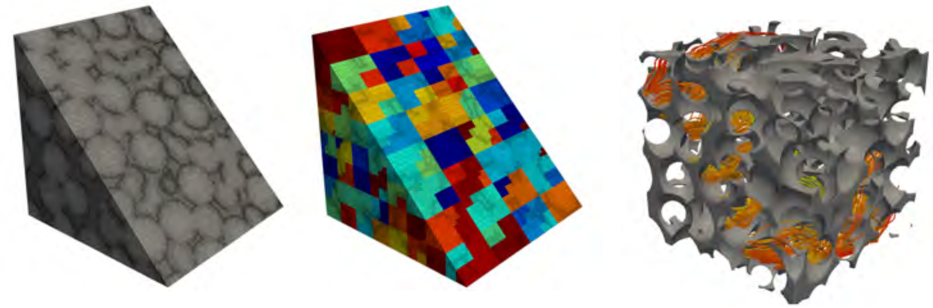


- Construct tree
- Upward pass – child-to-parent computations
- For each node – neighborhood far-field computations
- For each point/leaf node – near-field computations
- Downward pass – parent-to-child computations



(a) Upward Pass (b) Downward Pass

Scalability examples



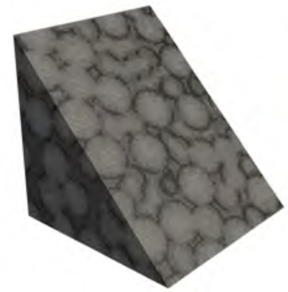
p	N_{dof}/p	N_{iter}	T_{solve}	TFLOPS	η
1	8.0E+6	155	477	0.36	1.00
6	7.8E+6	115	388	2.27	1.04
27	8.6E+6	101	401	10.3	1.05
125	8.5E+6	98	419	45.3	0.99
508	8.9E+6	92	444	173	0.94
2048	9.1E+6	90	474	656	0.88

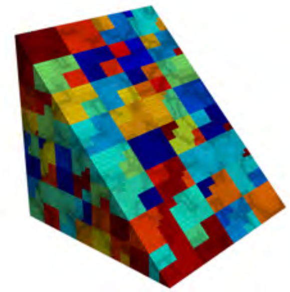
TACC's Stampede Xeon Phi + SandyBridge

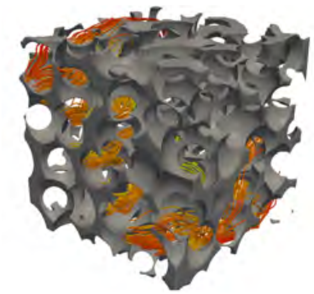
25% peak performance 35% peak LINPACK

Malhotra, Gholami, B., SC'14 *Best Student Paper Finalist*

Single-node







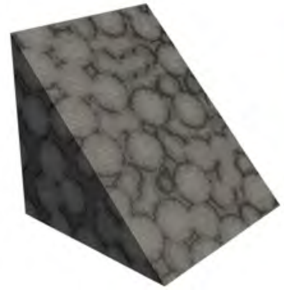
Single node performance: 16-core Xeon, Intel Phi

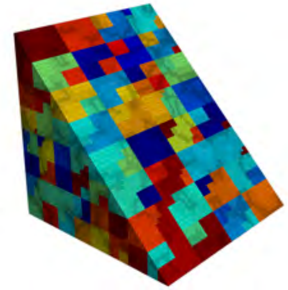
	Near		Far		All		
	T	GFLOPS	T	GFLOPS	T	GFLOPS	Speedup
Original	6.9	235	18.5	16	25.8	77	1.0
CPU Only	6.0	270	2.8	123	9.1	223	2.8
CPI+Phi	3.3	496	2.9	117	3.4	596	7.6

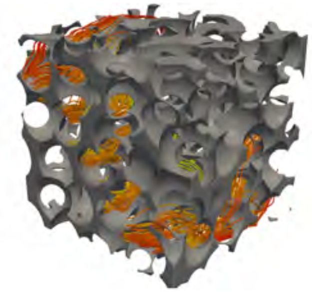
Algorithmic redesign; data structure redesign, blocking and loop reordering, OpenMP + AVX + SSE; prefetching; task parallelism, asynchronous offload

Malhotra, Gholami, B., SC'14 *Best Student Paper Finalist*

Hardware + algorithms







		CPU		CPU+Phi		CPU+GPU	
q	N_{oct}	T_{solve}	GFLOPS	T_{solve}	GFLOPS	T_{solve}	GFLOPS
8	32,768	1026.4	148	940.0	162	942.9	161
12	4,096	183.5	184	129.3	262	123.9	273
16	4,096	384.3	242	159.5	582	140.2	662
8	4,656	118.8	160	101.0	188	97.4	195
12	1,240	63.4	179	31.4	360	30.3	374
16	232	38.8	147	15.6	366	9.8	579

Malhotra, Gholami, B., SC'14 *Best Student Paper Finalist*

N-body methods: Scalability requirements

Computational physics/chemistry/biology

- large scale – 1E13 unknowns (3D/4D)

Graphics/signal analysis

- real time – 1E6 unknowns (1D/4D)

Data analysis/Uncertainty quantification

- large scale – 1E15 unknowns (1D - 1000D)
- real time / streaming / online

Computational kernels

Kernels

Geometric – distance calculations / projections

Analytic – special functions / fast transforms / differentiation

Algebraic – near / far field

Combinatorial – tree traversals / pruning / sorting / merging

Statistical – sampling

Memory – reshuffling, packing, permutation, communication

Application – linear/non linear/optimization/time stepping / multiphysics

Kernel examples **[problem size / node]**

- Special function evaluations
- Sorting/merging [1E5-1E9]
- Median/selection [1-1E9]
- Tree-traversals / neighborhood construction
- Euclidean distance calculations [GEMM-like 100—4000)]
- Nearest-neighbors [GEMM-like + heapsort]
- Kernel evaluations [GEMM-like 100—4000]
- FFTs [3D, size./dim O(10)-O(20)]
- High-order polynomial evaluations
- Rectangular GEMM [8-500 X 100000]
- Pivoted QR factorizations [500 – 4000]

MNIST – 8 M points / 784 dimensions

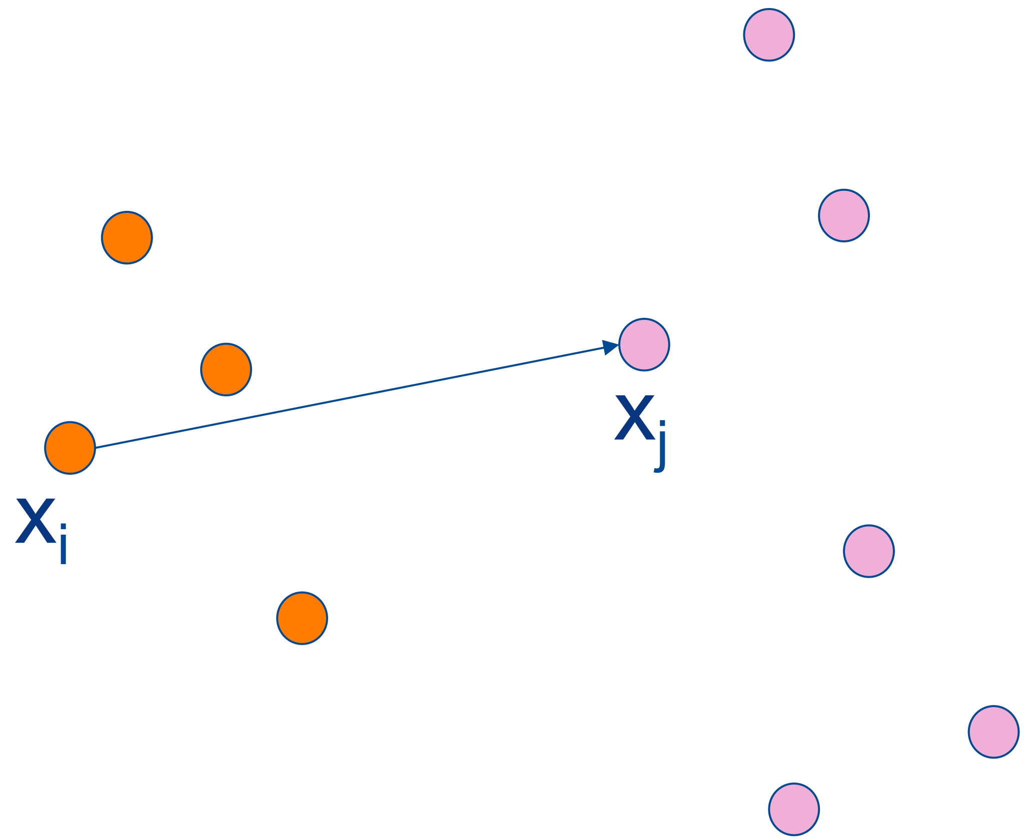
March, B. et al KDD'15

OCR using multiclass kernel logistic regression

#cores	**512**	**2,048**	**4,096**	**8,192**	**16,384**
Skel. (Alg. 2)	1,295	465	370	305	269
Lists (Alg. 3)	729	177	87	46	23
LET (Eq. 7)	273	136	107	87	71
Eval. (Eq. 14)	157	67	42	28	23
Total	2,471	862	621	483	394
Efficiency	1.00	0.72	0.50	0.32	0.20

Nearest neighbor kernel

$$\|x_i - x_j\|_2^2$$



$$= \|x_i\|_2^2 + \|x_j\|_2^2 - 2x_i^T x_j$$

Problem with using off-the-shelf GEMM

Input : O(Nd)

Output : O(N k)

Calculations: $O(N^2 d + N\,k \log k)$

Intermediate storage: $O(N^2 + Nk)$

Algorithm 2.1 `GEMM` Approach to k-Nearest Neighbor

$C = -2Q^T R$; `//GEMM(`-2`,`Q^T`,`R`,`0`,`C`);`

for each query i and reference j **do**

  $C(i,j) += Q_2(i) + R_2(j)$; `//`$-2x_i^T x_j + \|x_i\|_2^2 + \|x_j\|_2^2$`;`

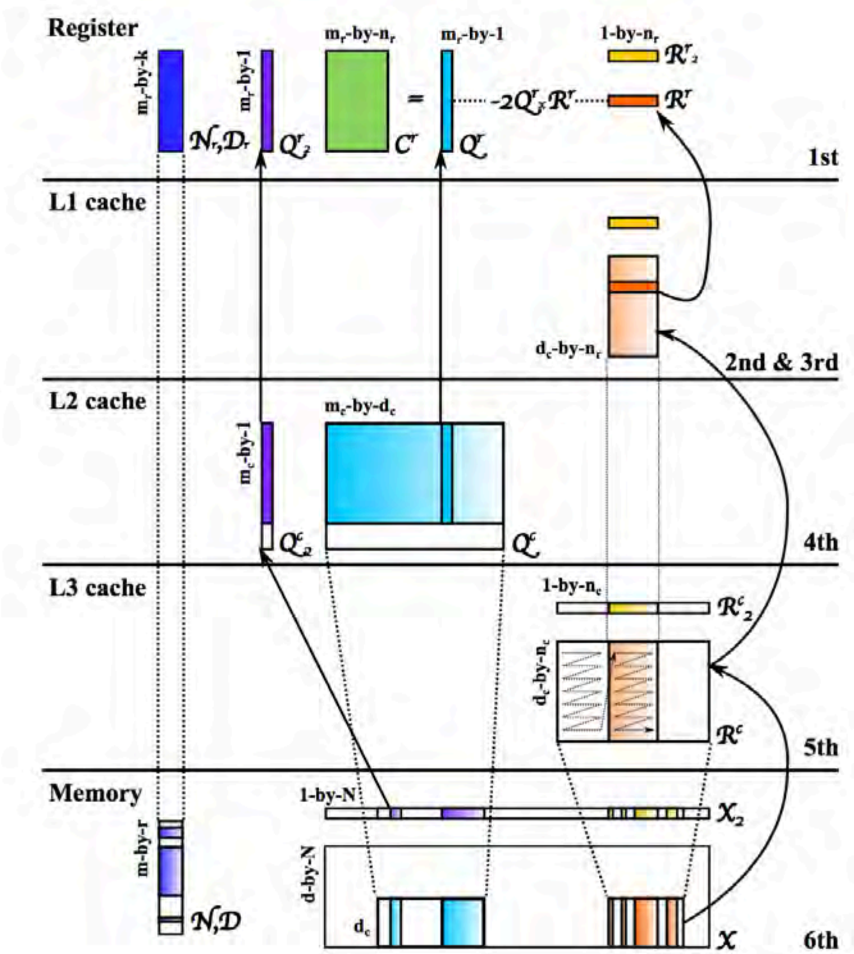
end for

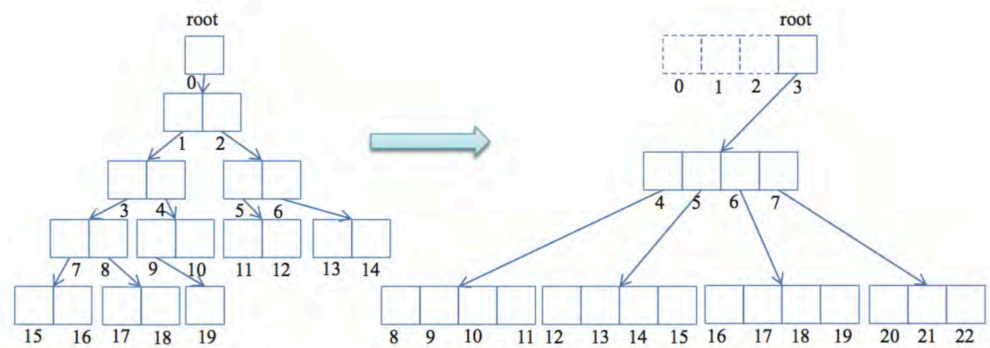
for each query i **do**

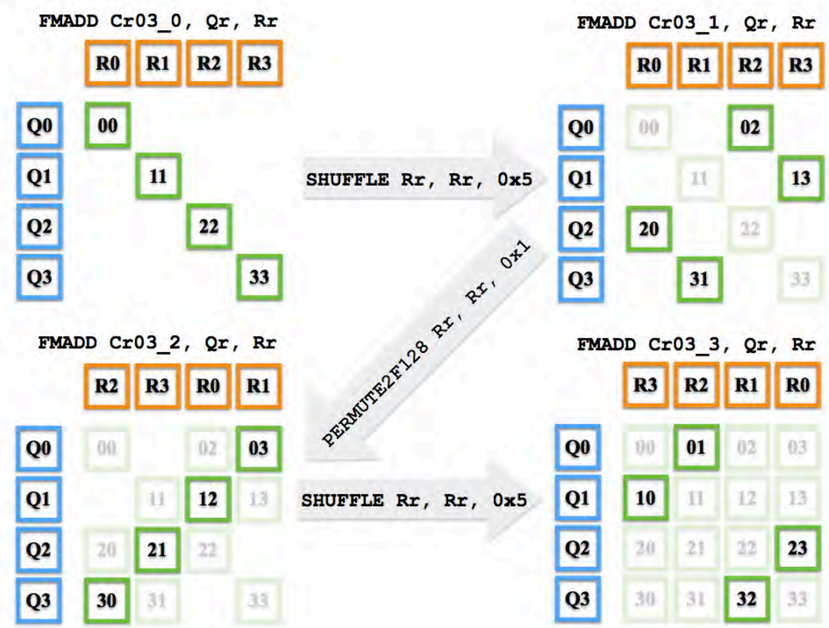
  Update $<\mathcal{N}(i,:),\mathcal{D}(i,:)>$ with $<r(:),C(i,:)>$

end for

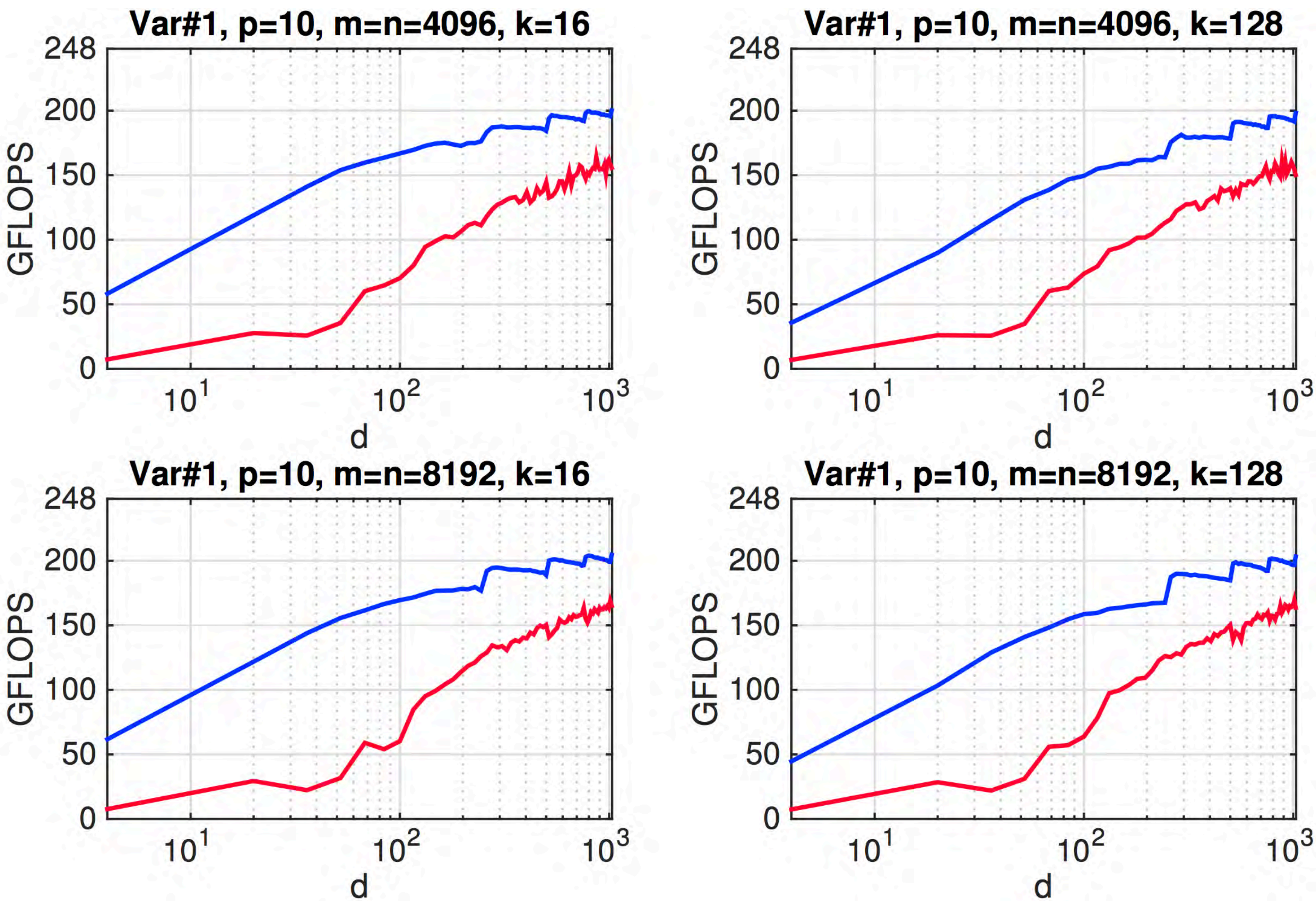
Custom GEMM + heap selection + arbitrary norms



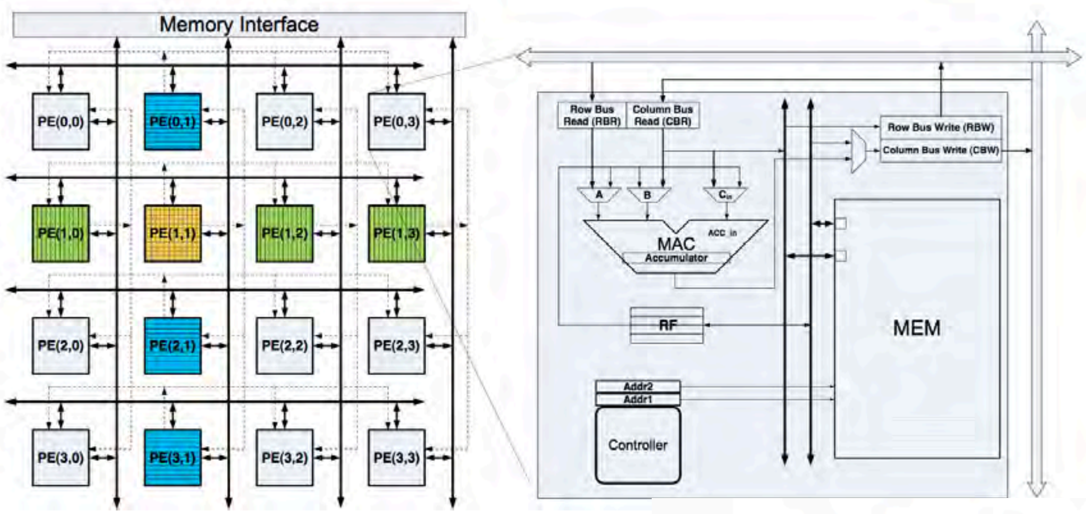


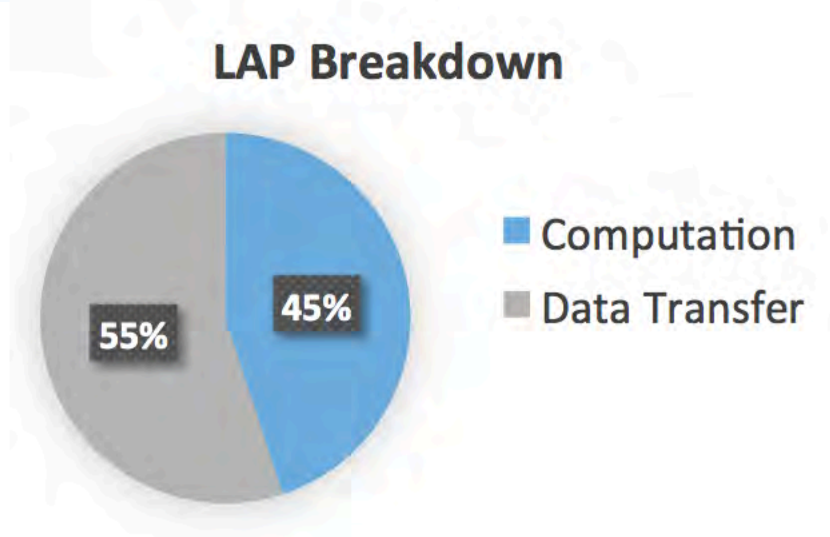


Comparison with GEMM



GEMM – like on special hardware





Linear Algebra Processor, A. Gertslauer UT Austin

The perspective of high-level application and library designers

actually just my perspective - focus on single node only

Workflow

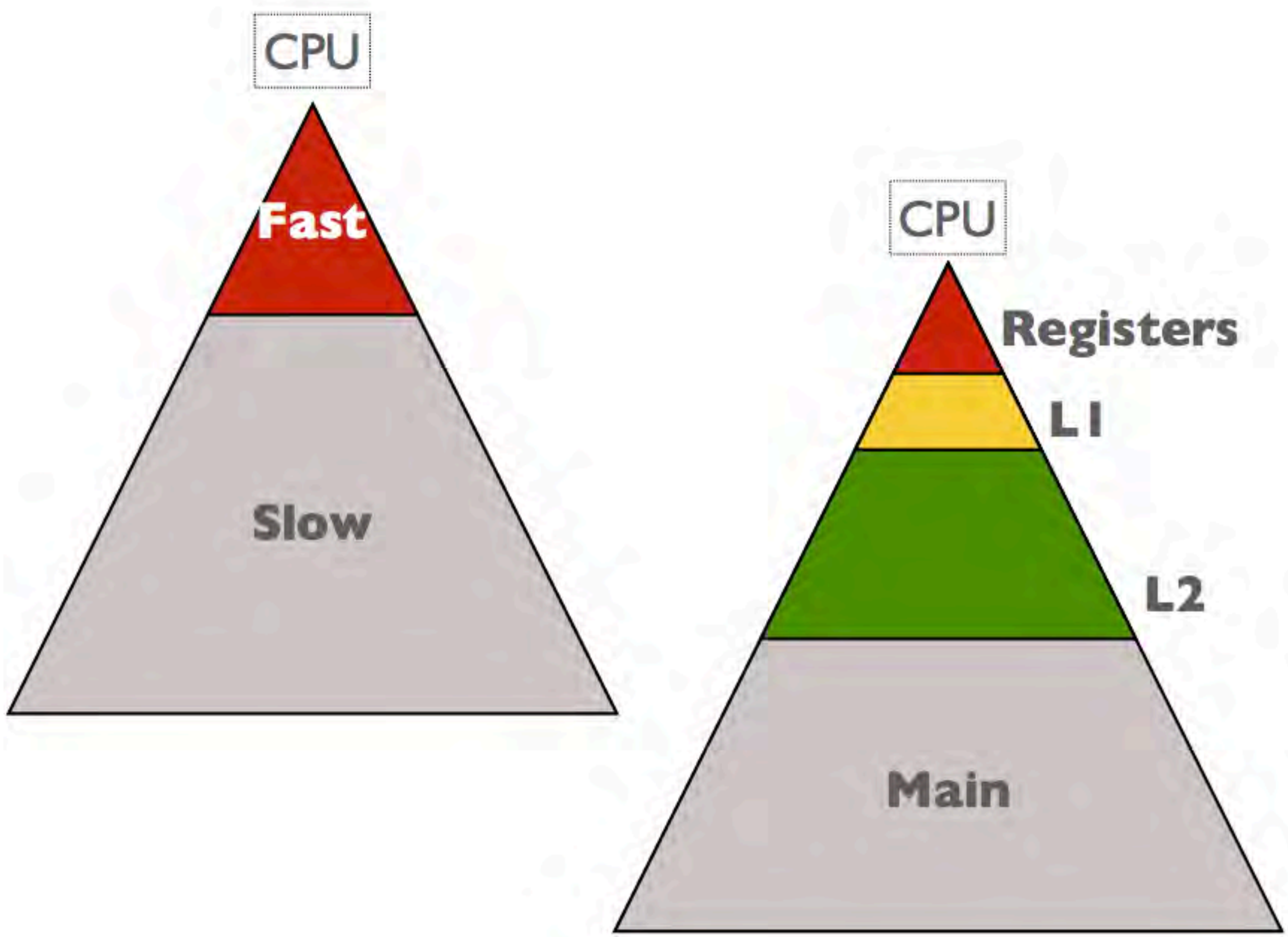
- Model selection
- Discretization - correctness and speed
- Verification
- Robustness / usability / reproducibility
- Validation
- **PREDICT**
- High performance and efficiency
  - Our responsibility: develop algorithms that “respect” memory hierarchies, promote/expose SIMD, are work-optimal, concurrent, and fault tolerant

Our understanding of CPU performance

Peak FLOPs =

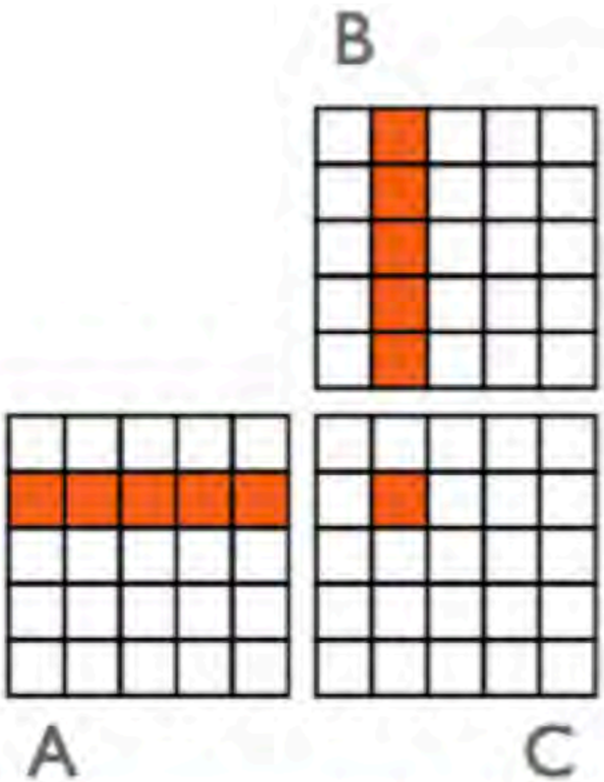
Number of cores ×	(Task-Level Parallelism)
FLOPS per instruction ×	(SIMD and FMA)
Instruction per cycle ×	(Instruction Level Parallelism)
Cycles per second	(Frequency)

Our understanding of memory hierarchy



Our coding preference

- Compute $C \leftarrow C + A \cdot B$




```
for i=1:n
  for j=1:n
    for k=1:n
      C(i,j) += A(i,k)*B(k,j)
```

4 lines of code

3 loops

The problem:

C=A*B, all A,B,C 4000-by-4000

1 CORE

Expected run time: $4000^3 * 2 / 21E9 =$ **6s**

ICC -g : 80 s, 12X slower

ICC -O: 48 s 8X slower

ICC -O3: 25 s; 4X slower

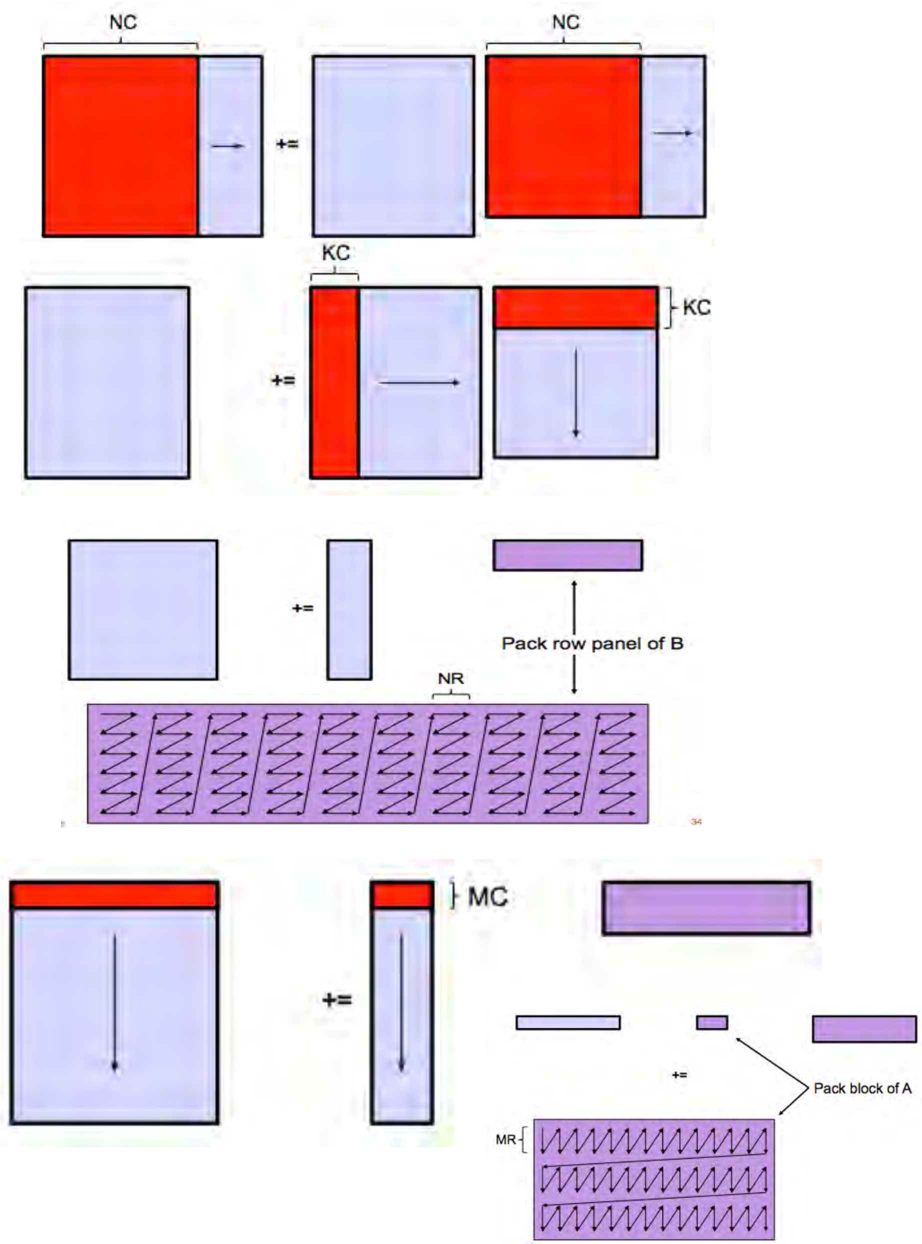
8 CORES: 3.4 s vs **0.75** (~4.5X slower)

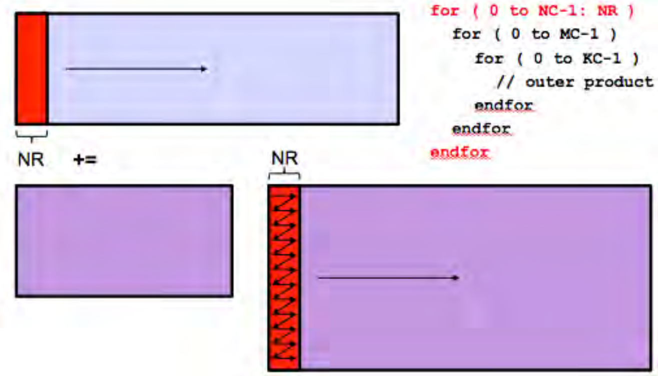
16 CORES: 2.6s vs **0.38** (~7X slower)

16 CORES, gcc -O3: 60secs (~100X slower)

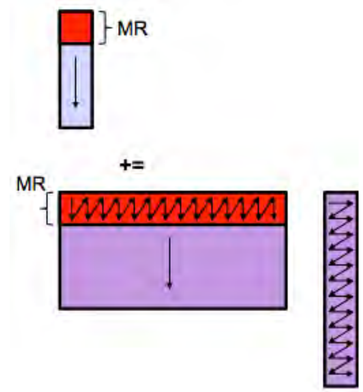
2 x 8core-E5-2680 2.7GHz

Reality

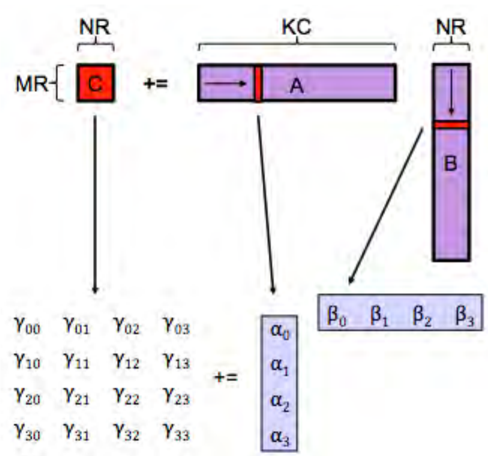





```
for ( 0 to NC-1: NR )
  for ( 0 to MC-1 )
    for ( 0 to KC-1 )
      // outer product
    endfor
  endfor
endfor
```




```
for ( 0 to NC-1: NR )
  for ( 0 to MC-1: MR )
    for ( 0 to KC-1 )
      // outer product
    endfor
  endfor
endfor
```




```
for ( 0 to NC-1: NR )
  for ( 0 to MC-1: MR )
    for ( 0 to KC-1: 1 )
      // outer product
    endfor
  endfor
endfor
```

- Typical micro-kernel loop iteration
  - Load column of packed A
  - Load row of packed B
  - Compute outer product
  - Update C (kept in registers)

Other applications that share same complexity

Geophysical applications

Plasma physics

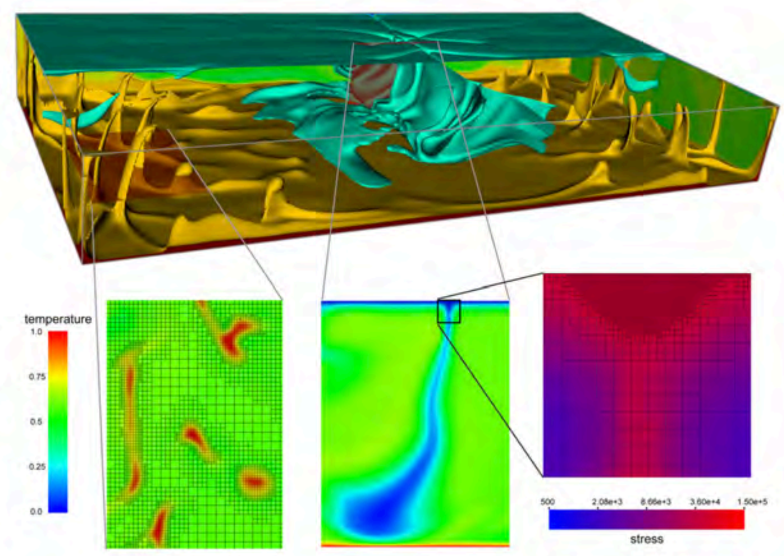
Materials/Biology/Chemistry

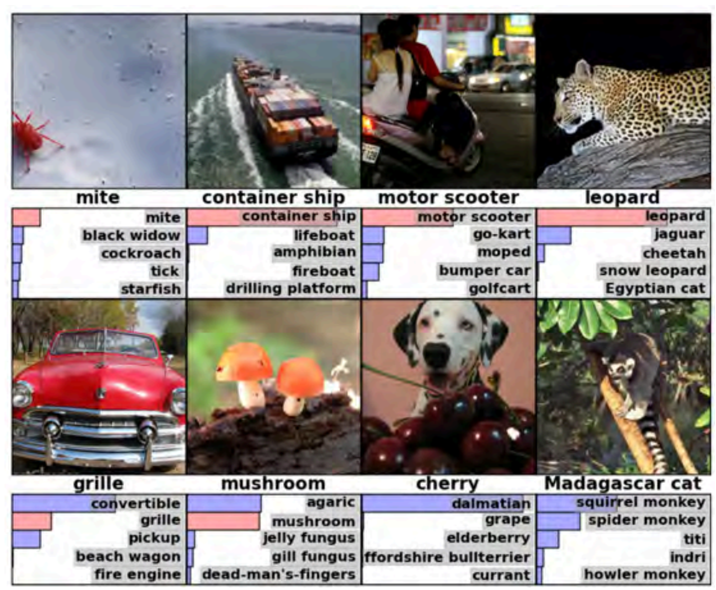
Inference

Design

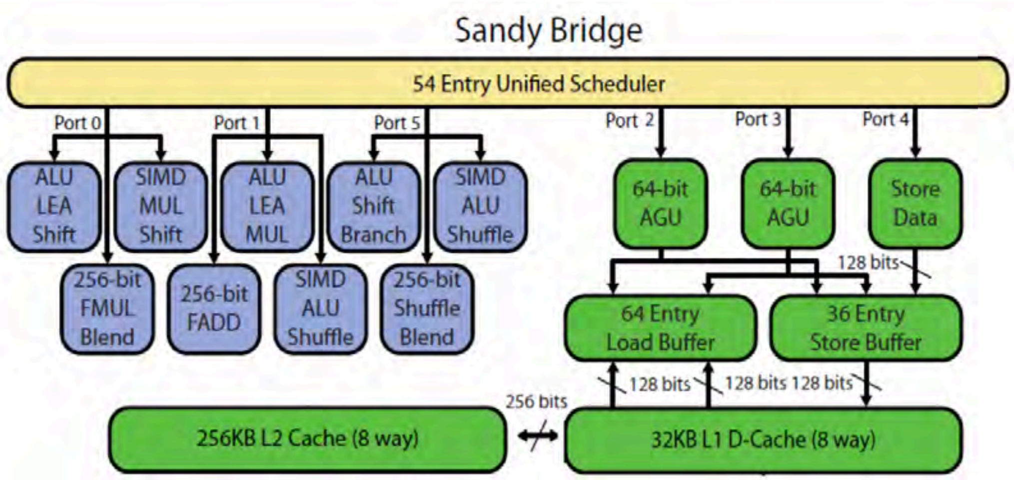
Data assimilation

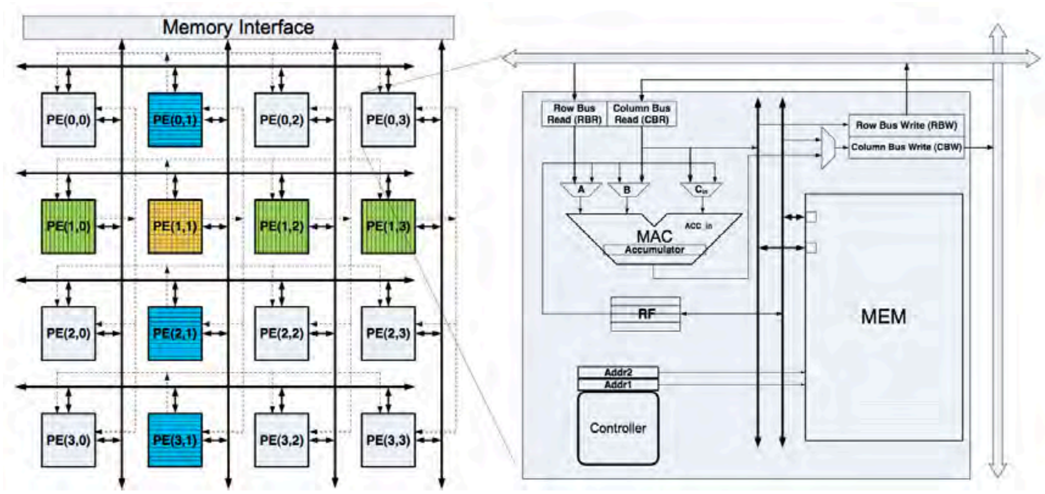
Machine learning





MPI + X: OpenMP, OpenCL, OpenACC, Pthreads, Intel TBB, CUDA, SSE/AVX





Challenges for HPC

Large and constantly evolving design space

Expanding complexity of simulation and data analysis

Expanding complexity of programming tools

Expanding complexity of underlying hardware

Expanding complexity of profiling tools

Absence of common abstract parallel machine model

Static/synchronous scheduling increasingly hard

Heterogeneous architectures

Fault tolerance / resiliency

End-to-end scalability

Gaps in

- programmability
- productivity
- maintenability
- performance
- scalability
- portability
- accessibility
- energy efficiency
- reproducibility

Large percentage of production scientific codes ~ 1-0.1% efficiency

Would be happy to have productivity tools that can raise performance to 10%

Requires 10X to 100X improvements

Wish list (besides faster hardware)

- Slower compilers
- Language support for multithreading, SIMD, and DAGs
- Memory hierarchy abstraction
- Language support for memory hierarchy
  - Async communication primitives (gather/scatter, reduce/broadcast, scans, all-to-all)
- "Eliminate" accelerators
- Standard library/language support
  - parallel I/O, special functions, sparse/dense linear algebra, trees, sort/search/scan/select

Future directions

Need more research on parallel algorithms

- algorithmic improvements have, typically, the largest impact

Need better abstractions for hierarchical memories and heterogeneous hardware

Need more stable and predictable programming APIs

Conclusions and summary

N-body codes

multiple kernels, data-structures

Algorithmic challenges

focus on data movement

Productivity challenges

Publications

Khawaja et al; ICPADS 14

Pedram et al; IEEE TC 14

Malhotra et al; SC 14; SISC 15;   ACM TOMS 15

March et al; SISC 15, IPDPS 15, KDD 15