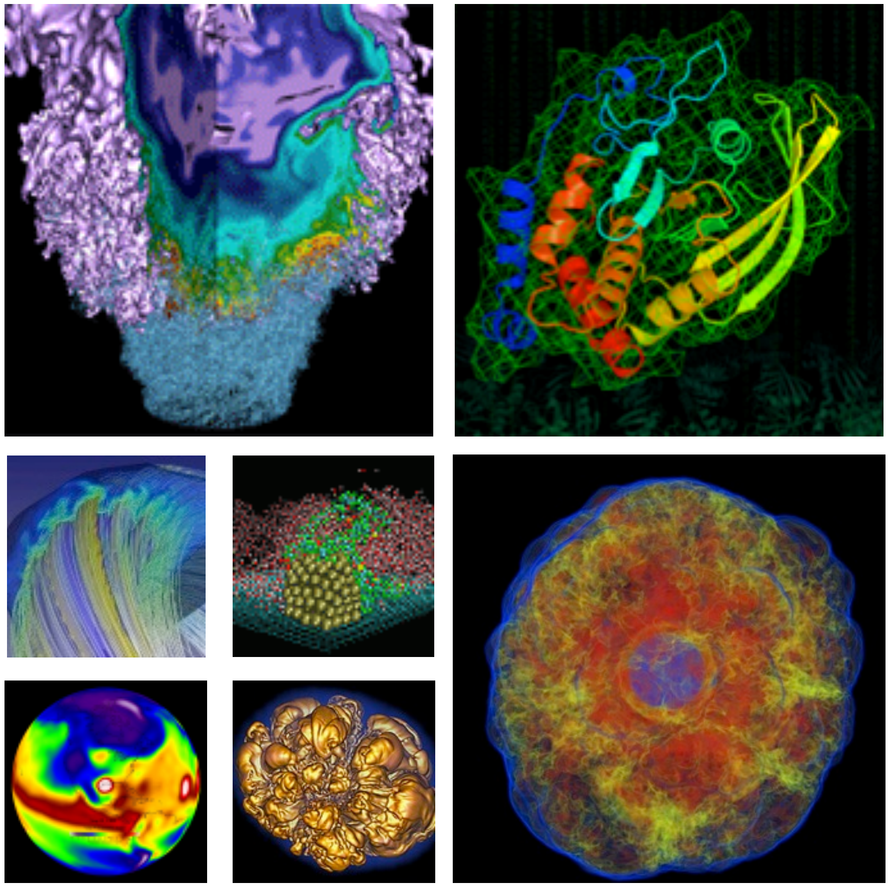
OpenMP Parallelization and Optimization of Graph-based Machine Learning Algorithms



Zhaoyi Meng, Alice Koniges, Yun (Helen) He,
Samuel Williams, Thorsten Kurth, Brandon Cook, Jack Deslippe, and
Andrea L. Bertozzi
University of California, Los Angeles
and
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory, USA

UCLA &

IWOMP Nara, Japan, October 5, 2016

OUTLINE

UCLA &

- Introduction to data classification and the new methods
- Semisupervised and unsupervised algorithms studied
- Optimization process
- Architecture testing
- Results and conclusions

Abstract

-New class of data clustering methods for segmentation of large datasets with graph based structure. The method combines ideas from classical nonlinear PDE-based image segmentation with fast linear algebra methods for computing information about the spectrum of the graph Laplacian.

-The goal of the algorithms is to solve semi-supervised and unsupervised graph cut optimization problems.

-Applications are to image processing such as image labeling and hyperspectral video segmentation

-We detail the OpenMP parallelization and algorithmic optimization in this presentation

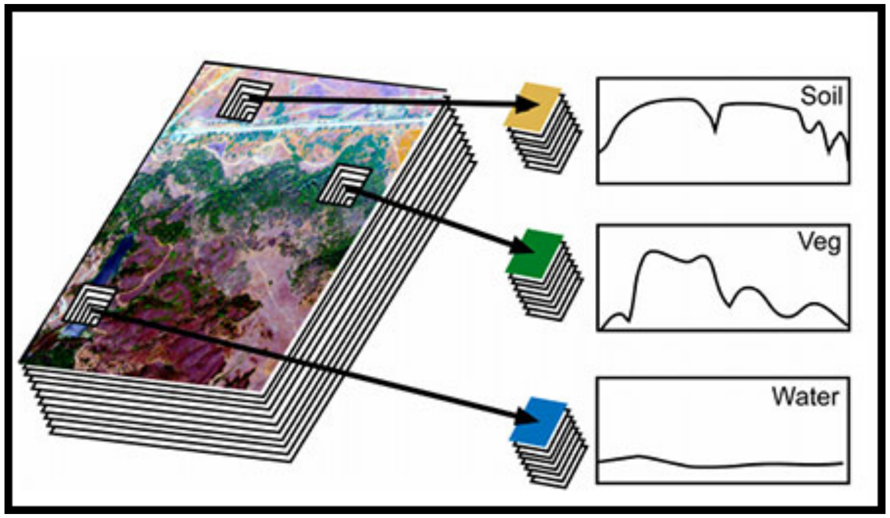
Novel UCLA Data classification algorithms are used to find similarities with improved accuracy.

UCLA & NERSC

Classify data sets (sort data into different classes), so that the similarity between nodes in one class is much larger than the similarity between nodes of different classes.

- Semi-supervised algorithm: have a small portion of known class labels.
- Unsupervised algorithm: have no knowledge of class labels at all.

Example: Classification of regions in a hyperspectral image of the earth

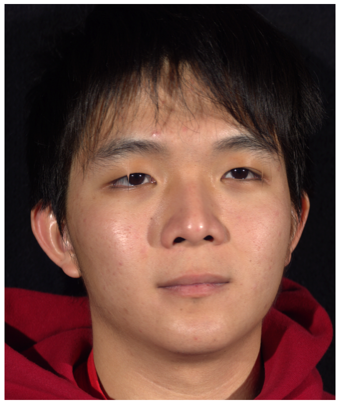


Each pixel contains many data channels

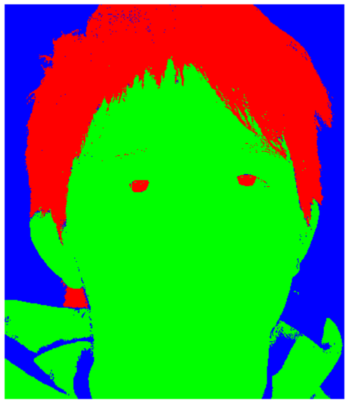
Two sample data sets (face and plume) and UCLA results using unsupervised algorithms

UCLA & NERSC

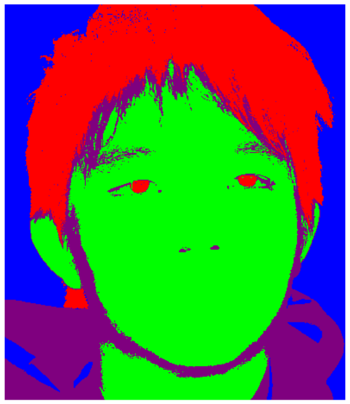
Ground truth



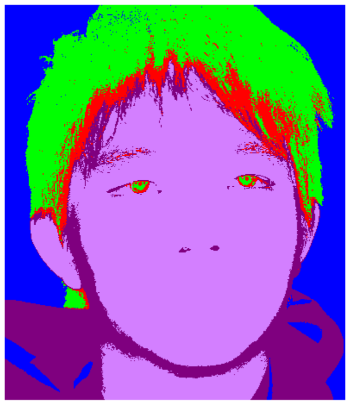
Three classes



Four classes

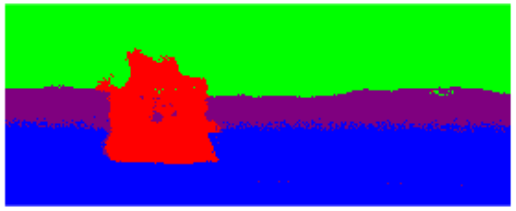


Five classes

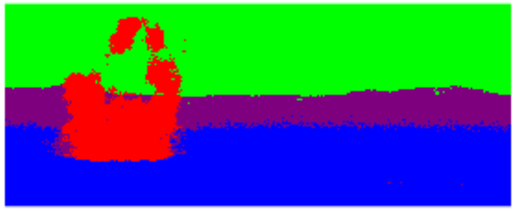


Data is 1372 ×1183 with 148 spectral bands (https://scien.stanford.edu/index.php/faces)

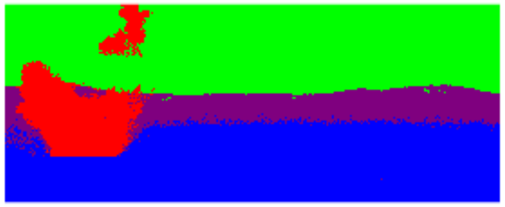
Time = 0 sec



Time = 15 sec



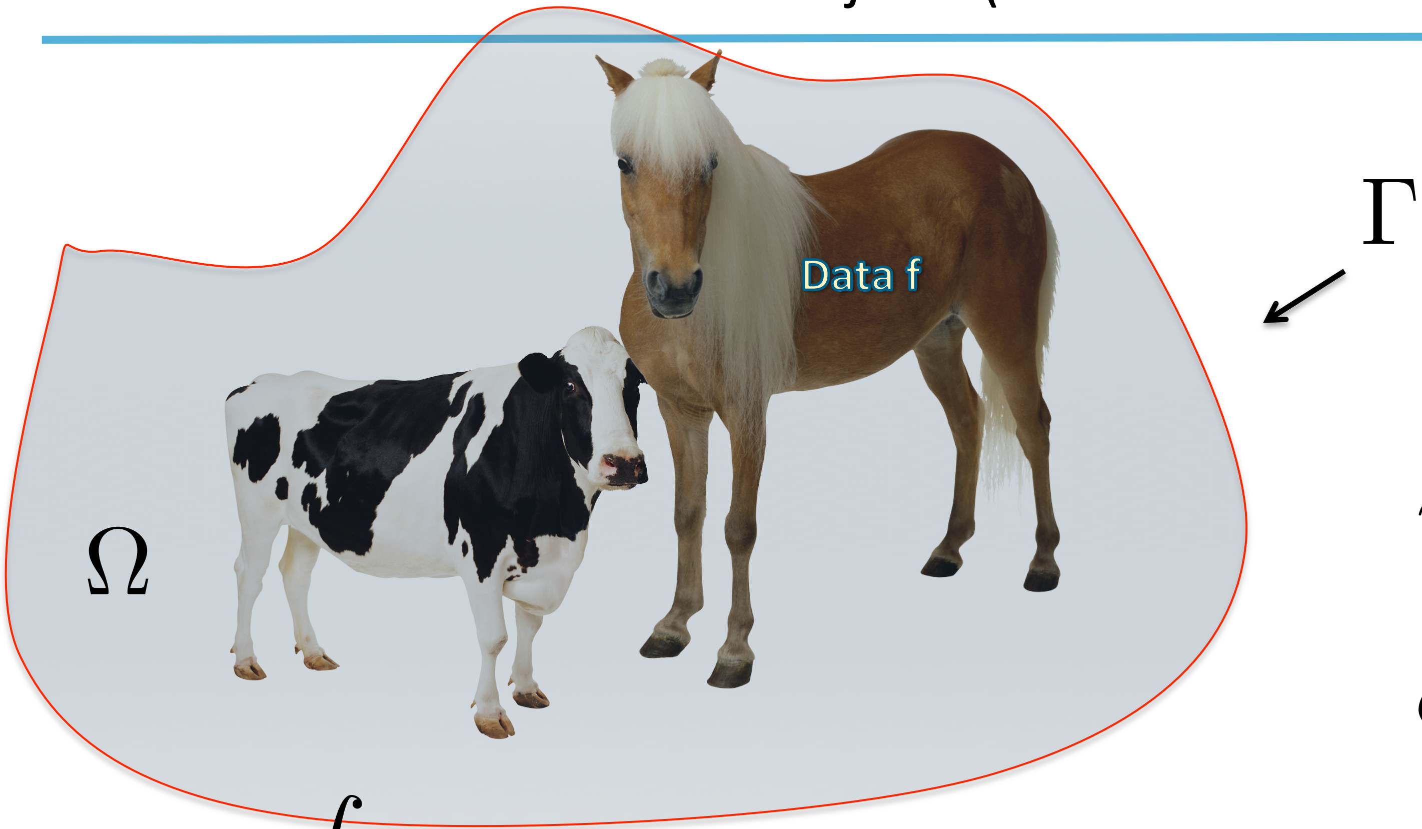
Time = 30 sec



Plume data is 128 × 320 with 129 infrared spectral bands and tracked using four classes (Chemical plumes released at the Dugway Proving Ground by J. B. Broadwater, et al.)

Method can be motivated by thinking about how to minimize curve around two objects (data classification)

UCLA & NERSC



Move curve to where there is lots of gradient structure in the image. Curve length is minimized. This process is connected to the total variation (TV). u is characteristic function of domain Ω

$$u = \chi_\Omega$$

$$\partial\Omega = \Gamma$$

$$|\Gamma| = \int |\nabla u| \equiv |u|_{TV}$$

$$\sim \frac{\epsilon}{2}\int |\nabla u|^2 + \frac{1}{\epsilon}\int W(u)dx \equiv GL_\epsilon(u)$$

Ginzburg-Landau functional

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

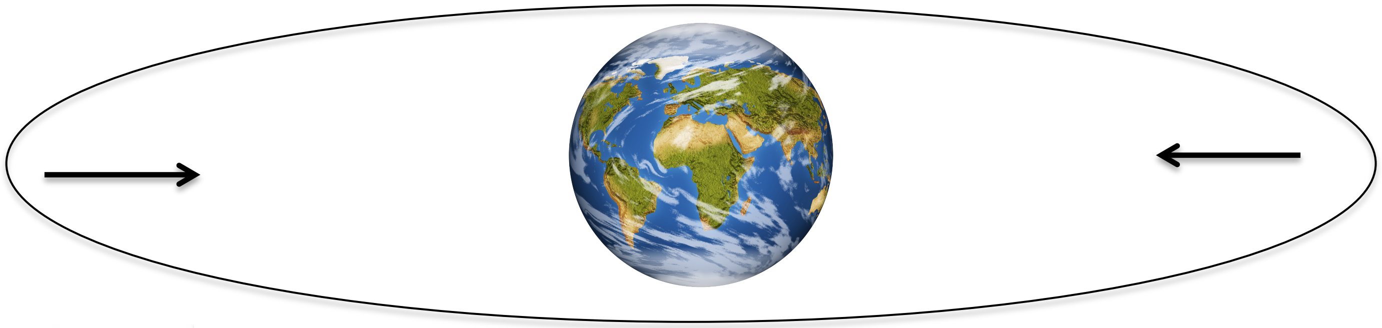
Diffuse interface Equations and their sharp interface limit

UCLA & NERSC

Gradient descent of GL function: $$u_t = \epsilon \Delta u - \frac{1}{\epsilon} W'(u)$$

First variation of the GL functional gives the Allen-Cahn equation. Famous in materials science. Now useful for data science.

$\epsilon \to 0$ Approximates Motion by Mean Curvature, steep sections move more.



U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

The two step MBO Scheme (1992) improves solution method performance

UCLA & NERSC

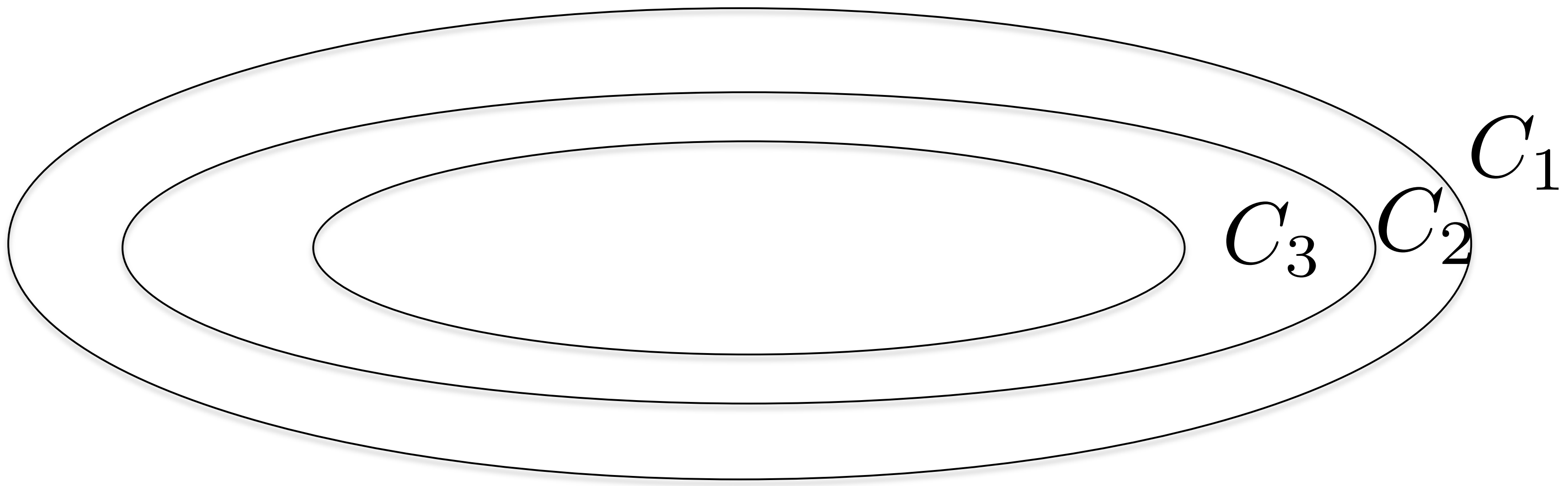
Merriman, Bence, Osher

$$\begin{cases} u_t - \Delta u = 0 \quad \text{in } (0,+\infty)\times\mathbb{R}^N, \\ u(0,x) = \begin{cases} 1, & x \in C_0, \\ -1, & x \in \mathbb{R}^N \backslash C_0. \end{cases} \end{cases} \longrightarrow \quad C_1 = \{x \in \mathbb{R}^N \mid u(h,x) \geq 0\}.$$

Heat equation

Threshold

iterate



Extended to Piecewise Constant Mumford-Shah Model by Esedoglu-Tsai 2006

PDE Methods were extended by Bertozzi et al. for similarity graphs for big data problems

UCLA &

PDE Motivated:
Euclidean Space Problem

$\downarrow$

- **Minimal surface problem**
- **Laplace operator**
- **Pseudo-spectral methods**
- **Fast Fourier Transform**
- **Uses all the modes**

Similarity Graphs for Large Data

$\downarrow$

- **Graph mincut problem**
- **Graph Laplacian**
- **Projection to eigensubspace of graph Laplacian**
- **Nystrom extension/ Rayleigh-Chebyshev**
- **Often only needs a small percentage of spectral modes.**

The basic UCLA algorithmic method is fast and accurate for a variety of data applications

UCLA & NERSC

I) Create a graph from the data, choose a weight function and then create the symmetric graph Laplacian.

II) Calculate the eigenvectors and eigenvalues of the symmetric graph Laplacian. *It is only necessary to calculate a portion of the eigenvectors*.*

III) Initialize assignment matrix u.

IV) Iterate the two-step scheme (MBO) until a stopping criterion is satisfied.

***Fast linear algebra routines are necessary – either Raleigh-Chebyshev procedure or Nystrom extension. We focus on the Nystrom version for this OpenMP optimization study.**

The workflow of the two algorithms we optimize is very similar.

UCLA & NERSC

- The major difference is that the semi-supervised algorithm requires fidelity (a small amount of "ground truth") while the unsupervised one does not. For very large data sets, we envision the unsupervised algorithm taking a dominant practical role.
- Step 1. Initialize parameters (total number of pixels, number of classes...) and read all frames of data file, fidelity(only for semi-supervised algorithm), initialize the labels of each pixel randomly.
- Step 2. Calculate eigenvectors and eigenvalues using a Nystrom scheme. Step 2a Parallelize time consuming parts of Nystrom via OpenMP.
- Step 3. Use Graph MBO or Graph Mumford-Shah algorithm to get labels of each pixel.
- Step 4. Output the labels of each pixel as classification result.

Unsupervised Graph MBO Algorithm

UCLA & NERSC

```
1. Input data matrix $f$, eigenvector matrix $\Phi$, eigenvalues $\{\lambda_k\}_{k=1}^{N}$.
2. Initialize $u^0$, $a^0 = \Phi^T \cdot u^0$
3. While $\frac{||u^{n+1}-u^n||_2^2}{||u^{n+1}||_2^2} < \alpha = 0.0000001$ do
   a. Updating $c$
         $c_k^{n+1} = \frac{<f, u_k^{n+1}>}{\sum_{i=1}^{N} u_{ki}}$
   b. Heat equation
         1. $a_k^{n+\frac{1}{2}} = a_k^n \cdot (1 - dt \cdot \lambda_k)$
         2. Calculating matrix $P$, where $P_{i,j} = ||f_i - c_j||_2^2$
         3. $y = \Phi \cdot a_k^{n+\frac{1}{2}} - dt \cdot \mu P$
   c. Thresholding
         $u_i^{n+1} = e_r, r = \arg\max_j y_i$
   d. Updating $a$
         $a^{n+1} = \Phi^T \cdot u^{n+1}$
```

Nystrom Extension Algorithm

- **In both the semi-supervised and unsupervised algorithms, we calculate the leading eigenvalues and eigenvectors of the graph Laplacian using the Nystrom method to accelerate the computation**
- **This is achieved by calculating an eigendecomposition on a smaller system of size M << N and then expanding the results back up to N dimensions. The computational complexity is almost O(N).**

$$W = \begin{pmatrix} W_{XX} & W_{XY} \\ W_{YX} & W_{YY} \end{pmatrix}, \quad W \sim \begin{pmatrix} W_{XX} \\ W_{YX} \end{pmatrix} W_{XX}^{-1} \begin{pmatrix} W_{XX} & W_{XY} \end{pmatrix}.$$

Computing $W_{XX}, W_{XY} = W_{YX}^T$ requires only $(|X| \cdot (|X| + |Y|)$ computations versus $(|X| + |Y|)^2$ for the whole similarity matrix. The method approximates W_{YY} by $W_{YX} W_{XX}^{-1} W_{XY}$ and the error is determined by how much the rows of W_{XY} span the rows of W_{YY}.

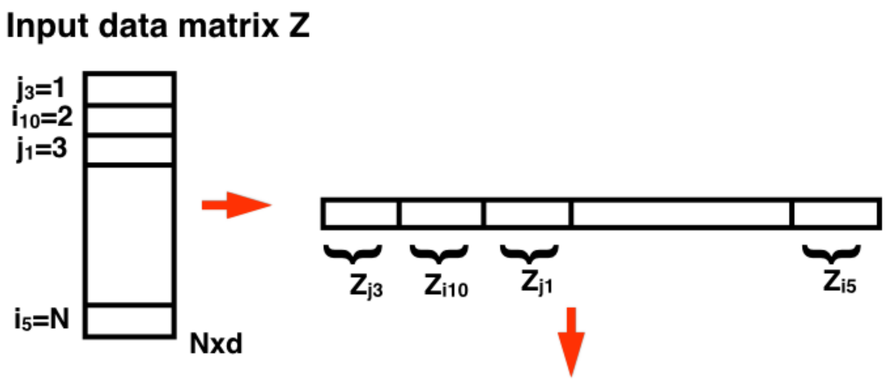
To start the optimization procedure we find hot spots and use libraries where possible

UCLA &

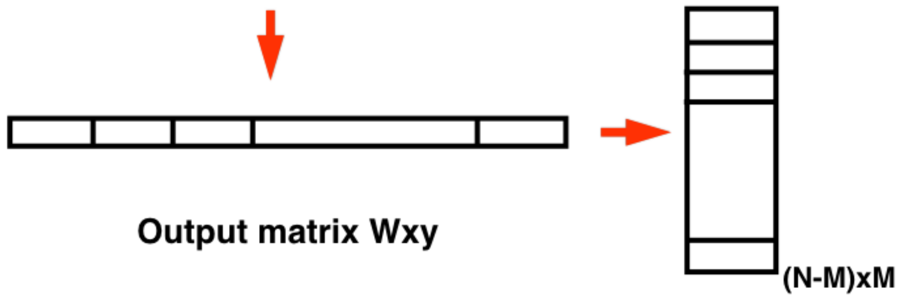
- Data are in matrix form and require intensive linear algebra calculations.
- Use LAPACK (Linear Algebra PACKage) and BLAS (Basic Linear Algebra Subprograms).
  - Use of BLAS 3 (matrix - matrix) instead of BLAS 1 (vector – vector) yields better performance
- VTune analysis – calculating Wxy takes 90% of the runtime of the Nystrom extension – a good candidate for OpenMP parallelization.

The matrix formed by calculating Wxy in the Nystrom extension takes 90% of the runtime

UCLA & NERSC



```
#pragma omp parallel for
    for(j = 0; j<N-M; j++)
    {
        n2 = <Z_j,Z_j>
        for(int i = 0; i<M; i++){
            n12 = <Z_i,Z_j>;
            n1 = <Z_i,Z_i>;
            d = 1-n12/sqrt(n1*n2);
            Wxy[j*m+i]=exp(-d/sigma);
        }
    }
```



U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Optimization of the Nystrom Loop

UCLA &

NERSC

Analysis with VTune shows that the construction of W_{XY} is the most time consuming phase

We investigate four steps to evaluate performance/optimization

Step A: parallelizing the inner j-loop and BLAS3 optimization on Graph MBO.

Step B: parallelizing the outer j-loop.

Step C: normalizing and forming all Z_js to X_{mat}.

Step D: using uniform sampling and chunked Y matrices.

An additional speed up comes from reordering loops

UCLA &

Step A: Parallelizing the inner j-loop

```
for $i = 0; i < M; i++$
    $n_1 = < Z_i, Z_i >$
    #pragma omp parallel for
    for $j = 1 : N - M$
        $n_{12} = < Z_i, Z_j >$
        $n_2 = < Z_j, Z_j >$
        $d = 1 - n_{12}/\sqrt{n_1 \cdot n_2}$
        $W_{XY}(i,j) = exp(-d/\sigma)$
    end
end
```

Step B: Parallelizing the outer j-loop

```
#pragma omp parallel for
for $j = 1 : N - M$
    $n_2 = < Z_j, Z_j >$
    for $i = 1 : M$
        $n_{12} = < Z_i, Z_j >$
        $n_1 = < Z_i, Z_i >$
        $d = 1 - n_{12}/\sqrt{n_1 \cdot n_2}$
        $W_{XY}(i,j) = exp(-d/\sigma)$
    end
end
```

Use Matrix Form and BLAS 2

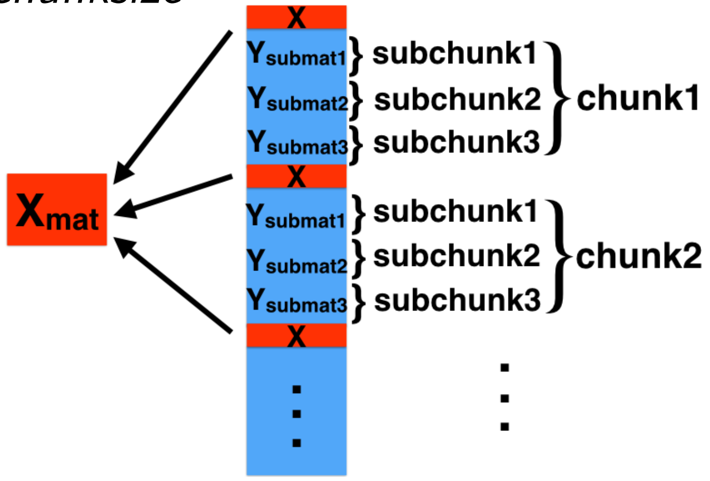
Step C: Calculating W_{XY}, normalize and form all Z_is to X_{mat}

```
#pragma omp for
for $j = 1 : N - M$
    $n_2 =< Z_j, Z_j >$
    $n_{vec} = 1- < X_{mat}, Z_j > /\sqrt{n2}$
    #pragma omp simd aligned
    for $i = 1 : M$
        $W_{XY}(i, j) = exp(-n_{vec}/\sigma)$
    end
end
```

We can form chunks to further optimize

Step D: Calculating W_{XY} using uniform sampling and chunked Y matrices

```
#pragma omp for collapse(2)
for $ychunk = 0; ychunk < m; ychunk++$
    for $j = chunkstart; j < chunkstop; j += subchunksize$
        #pragma omp simd aligned
        for $k = 0; k < subchunksize; k++$
            $n2_{vec}[k] = \langle Z_{j+k}, Z_{j+k} \rangle$
            $n2_{vec}[k] = 1/\sqrt{n2_{vec}[k]}$
        end
        $n12_{mat} = \langle X_{mat}, Y_{submat_j} \rangle$
        #pragma omp simd aligned
        for $i = 0; i < m; i++$
            for $k = 0; k < subchunksize; k++$
            $d = 1 - n12_{mat}[i,k] \cdot n2_{vec}[k]$
            $W_{XY}(i, j+k) = exp(-d/\sigma)$
        end
```



Choosing the sub-chunk size. Too small, wastes potential of combining expensive operations. If it is too large, the sub-chunk may run out of lower level cache and needs to be put into the higher cache levels, up to the point where they spill over into DRAM which may cause a substantial performance hit. The optimal value depends on the cache hierarchy, their respective sizes, their latency and so on.

Thread affinity settings also affect performance

UCLA &

- We choose the thread affinity setting as “OMP_PROC_BIND = spread” and
  “OMP_PLACES = cores/threads”,
  because it uses one hardware thread per core.
- While if we use
  “OMP_PROC_BIND = close” and “OMP_PLACES = threads”,
  it puts more threads on each physical core and leaves other cores idle, which affects scaling performance.

Testing of algorithms is done on Cori Phase I and KNL White Boxes

UCLA & NERSC

Cray XC40 with Knights Landing

Cori Phase I:
Cray XC based on the Intel Haswell multi-core processor. Each node has 128GB of memory and two 2.3 GHz 16-core Haswell processors. Each core has its own L1 and L2 caches, with 64 KB and 256 KB, respectively.

Knight's Landing (KNL) Many Integrated Core (MIC) Architecture: The system has 64 cores with 1.3 GHz clock frequency. Each core has two 512 bit-wide vector processing units. 16GB on package memory are shared between all cores. The 512 KB L2 cache is shared between two cores and 16 KB L1 cache is private to the core.

Theoretical Peak performance: Phase I Haswell: 1.92 PFlops/sec; Phase II KNL: 27.9 PFlops/sec.

- Total compute nodes: Phase I Haswell: 1,630 computes nodes, 52,160 cores total (32 cores per node);

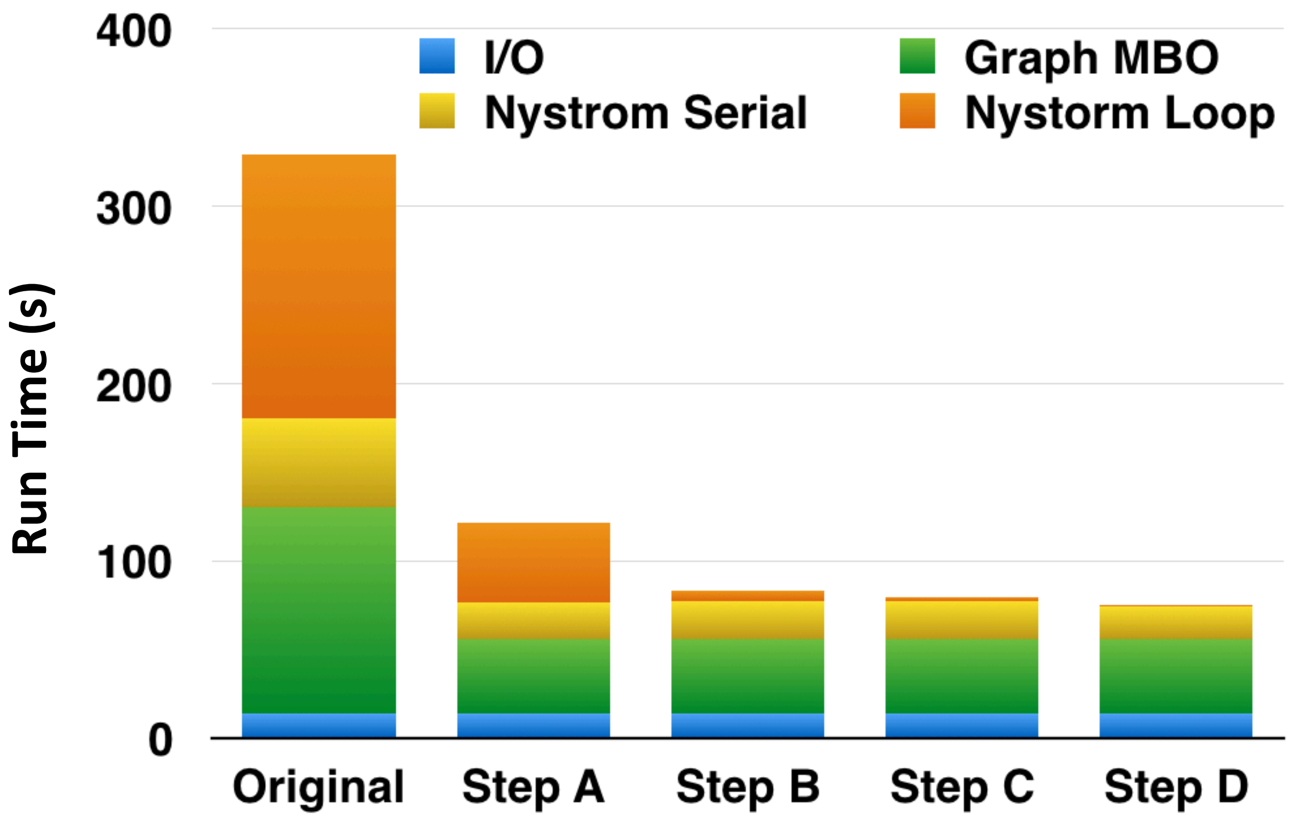
Phase II KNL: 9,304 compute nodes, 632,672 cores in total (68 cores per node).

- Cray Aries high-speed interconnect with Dragonfly topology (0.25 µs to 3.7 µs MPI latency, ~8GB/sec MPI bandwidth)
- Aggregate memory: Phase I Haswell partition: 203 TB; Phase II KNL partition: 1 PB.
- Scratch storage capacity: 30 PB

(Open for first testing this morning !!)

Optimization Results

UCLA &



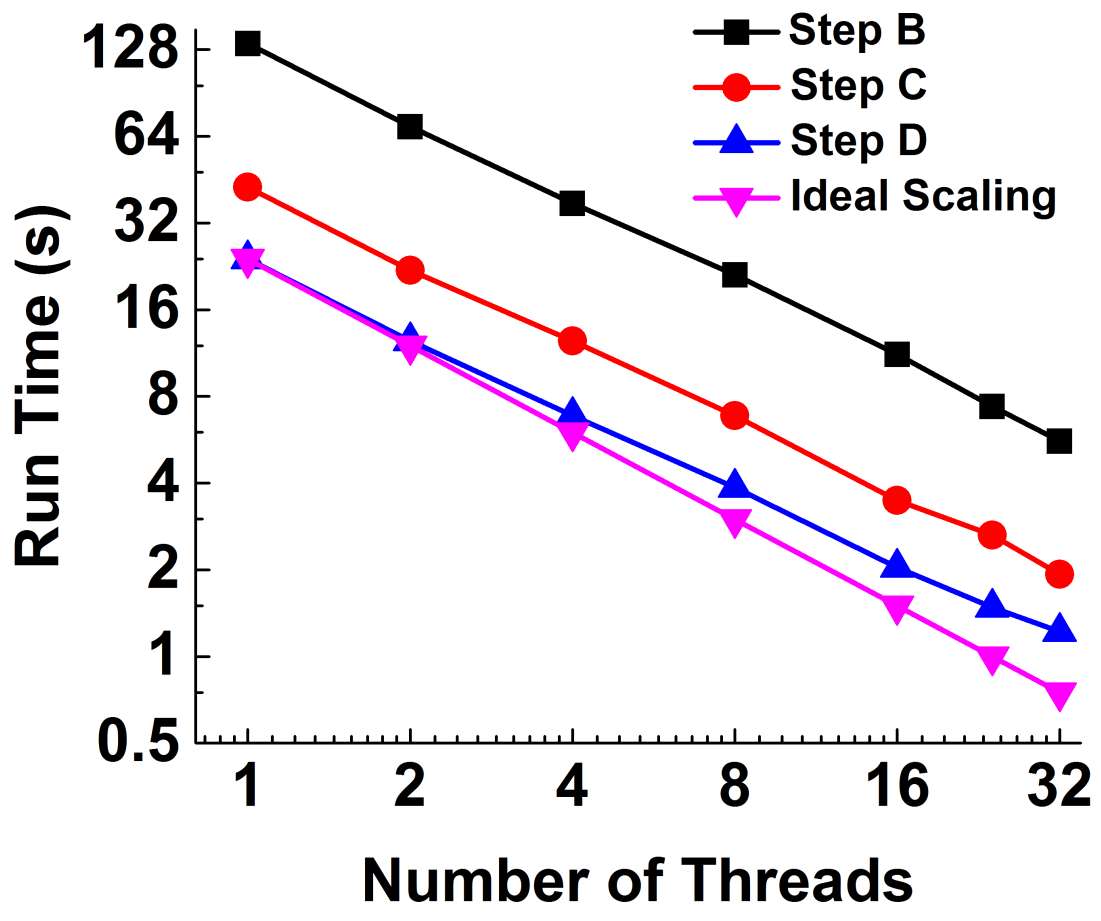
Step A: parallelizing the inner j-loop and BLAS3 optimization on Graph MBO.

Step B: parallelizing the outer j-loop.

Step C: normalizing and forming all Z_is to X_{mat}.

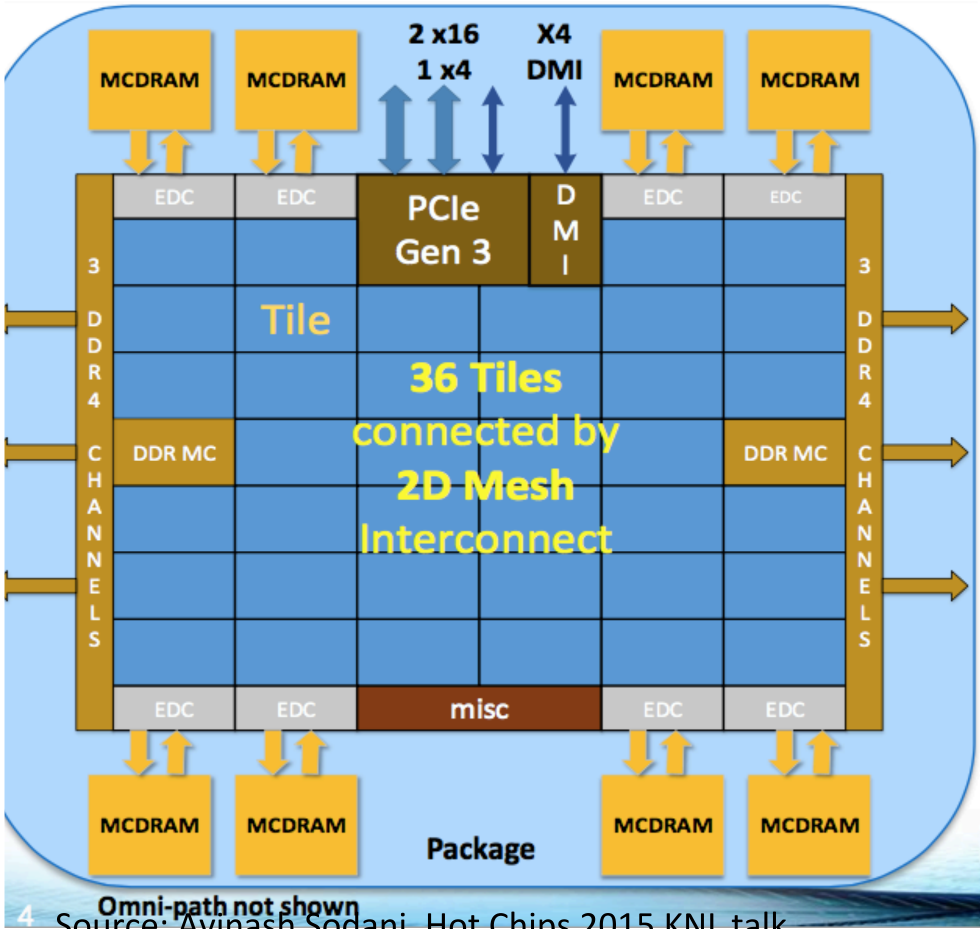
Step D: using uniform sampling and chunked Y matrices.

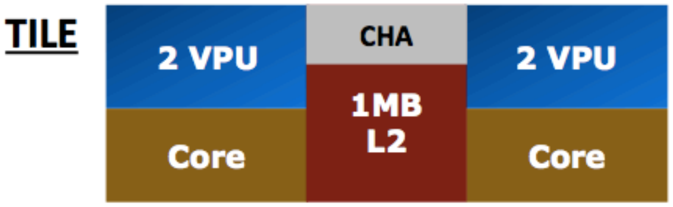
Nystrom Loop Scaling: Cori Phase 1 Results



KNL (our white boxes) are latest from Intel with lots of cores with in package memory

Knights Landing Overview





Chip: 36 Tiles interconnected by **2D Mesh**
Tile: 2 Cores + 2 VPU/core + 1 MB L2

Memory: MCDRAM: 16 GB on-package; High BW
DDR4: 6 channels @ 2400 up to 384GB
IO: 36 lanes PCIe Gen3. 4 lanes of DMI for chipset
Node: 1-Socket only
Fabric: Omni-Path on-package (not shown)

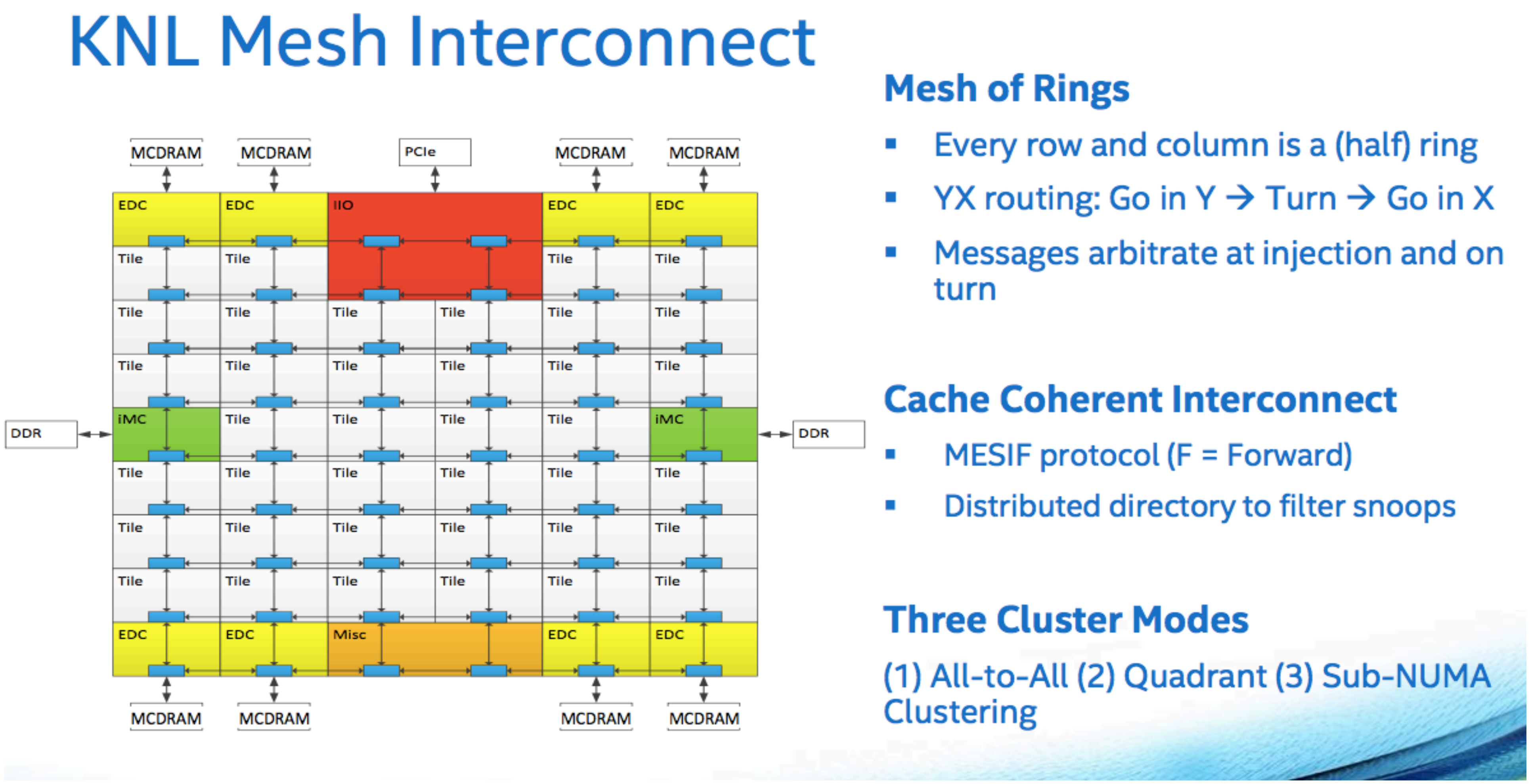
Vector Peak Perf: 3+TF DP and 6+TF SP Flops
Scalar Perf: ~3x over Knights Corner
Streams Triad (GB/s): MCDRAM : 400+; DDR: 90+

Source Intel: All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice. KNL data are preliminary based on current expectations and are subject to change without notice. 1Binary Compatible with Intel Xeon processors using Haswell Instruction Set (except TSX). [2]Bandwidth numbers are based on STREAM-like memory access pattern when MCDRAM used as flat memory. Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

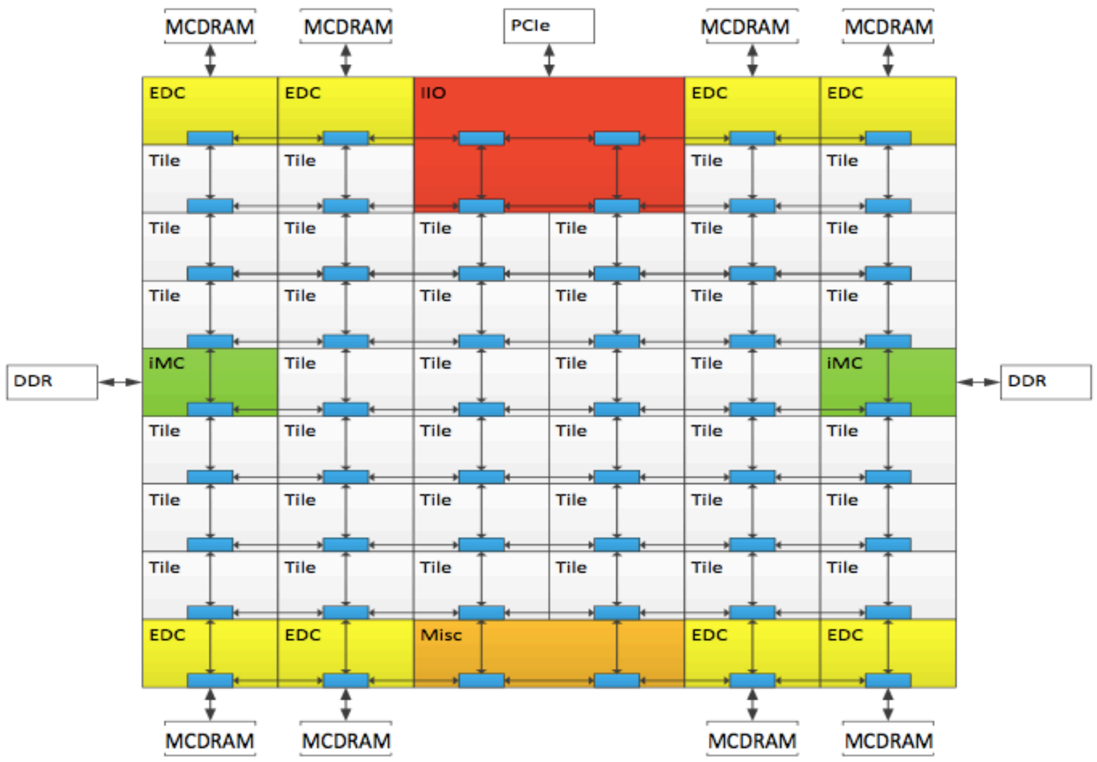
4

Source: Avinash Sodani, Hot Chips 2015 KNL talk

Connecting tiles



KNL Mesh Interconnect



Mesh of Rings

- Every row and column is a (half) ring
- YX routing: Go in Y → Turn → Go in X
- Messages arbitrate at injection and on turn

Cache Coherent Interconnect

- MESIF protocol (F = Forward)
- Distributed directory to filter snoops

Three Cluster Modes

(1) All-to-All (2) Quadrant (3) Sub-NUMA Clustering

Source: Avinash Sodani, Hot Chips 2015 KNL talk

Network interface Chip in the package …

KNL w/ Intel® Omni-Path

Omni-Path Fabric integrated *on package*

First product with integrated fabric

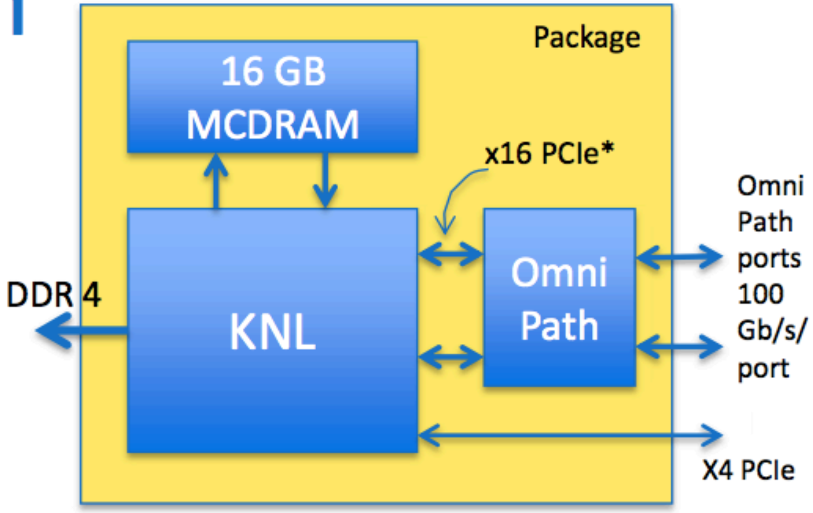
Connected to KNL die via 2 x16 PCIe* ports

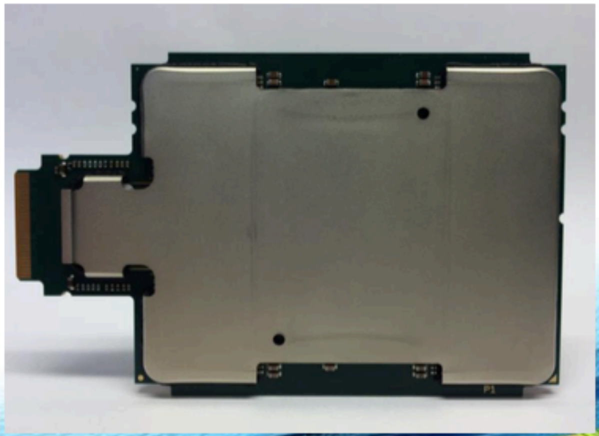
Output: 2 Omni-Path ports

- 25 GB/s/port (bi-dir)

Benefits

- Lower cost, latency and power
- Higher density and bandwidth
- Higher scalability





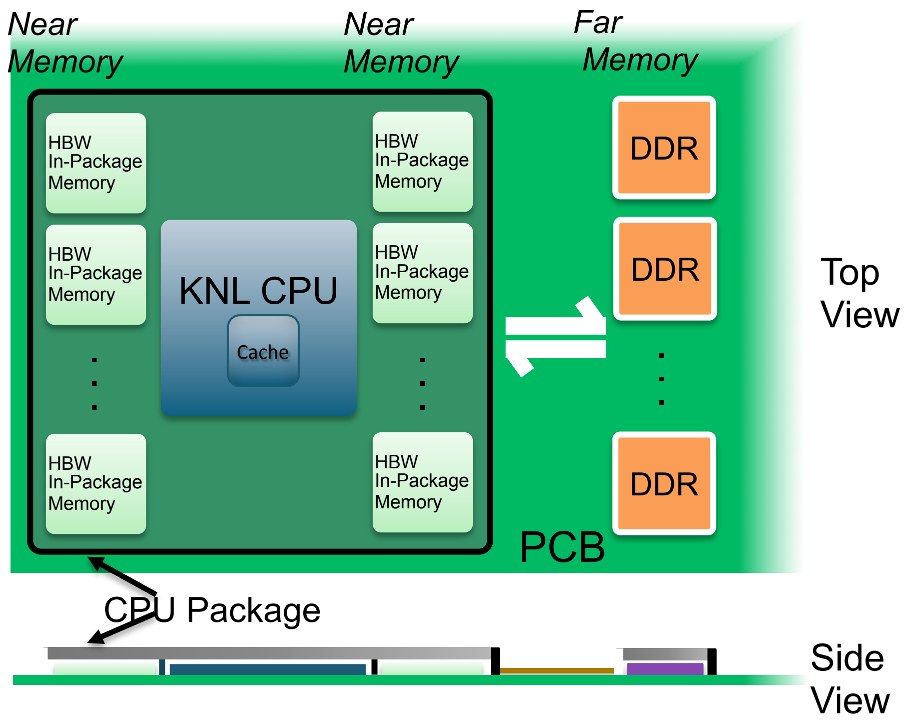
*On package connect with PCIe semantics, with MCP optimizations for physical layer

Source: Avinash Sodani, Hot Chips 2015 KNL talk

Knights Landing Integrated On-Package Memory

UCLA & NERSC

Cache Model	Let the hardware automatically manage the integrated on-package memory as an "L3" cache between KNL CPU and external DDR
Flat Model	Manually manage how your application uses the integrated on-package memory and external DDR for peak performance
Hybrid Model	Harness the benefits of both cache and flat models by segmenting the integrated on-package memory

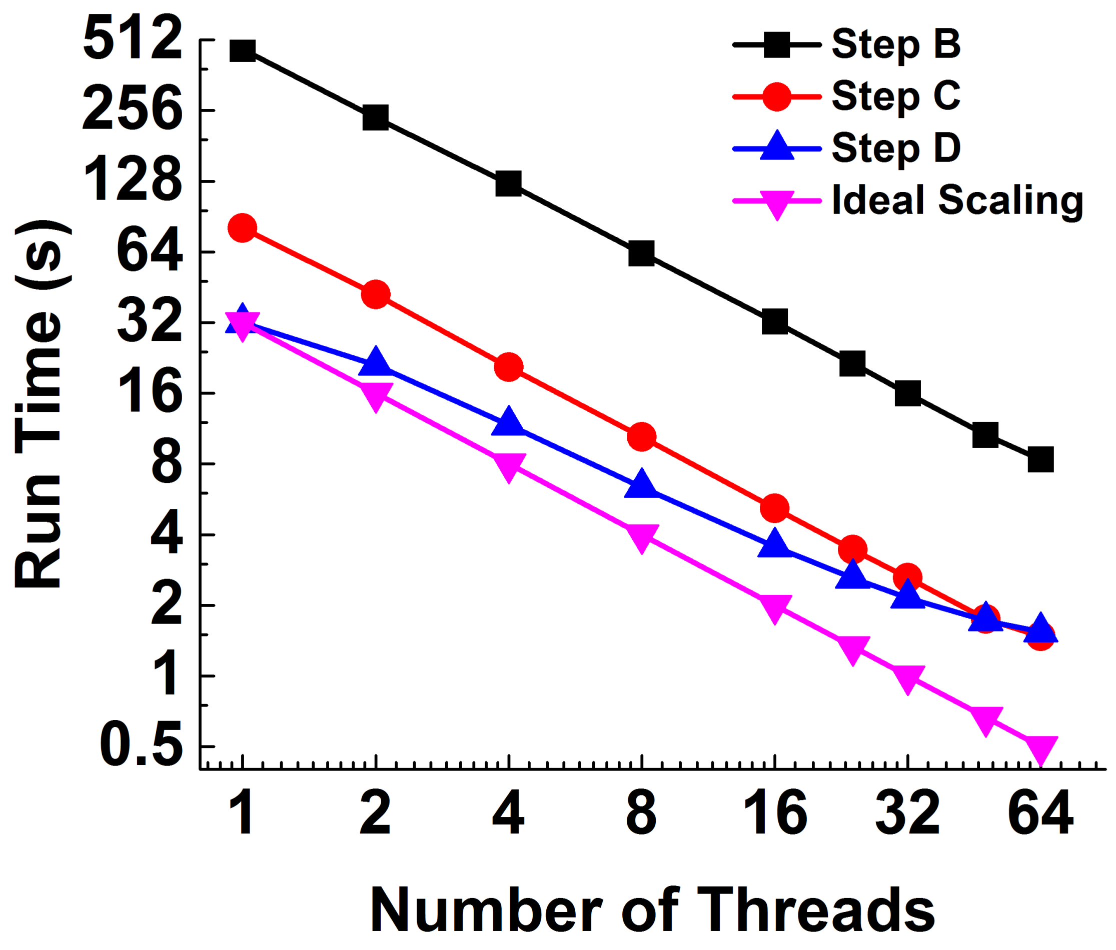


Maximum performance through higher memory bandwidth and flexibility

Slide from Intel

UCLA & NERSC

Nystrom Loop Scaling: KNL Results



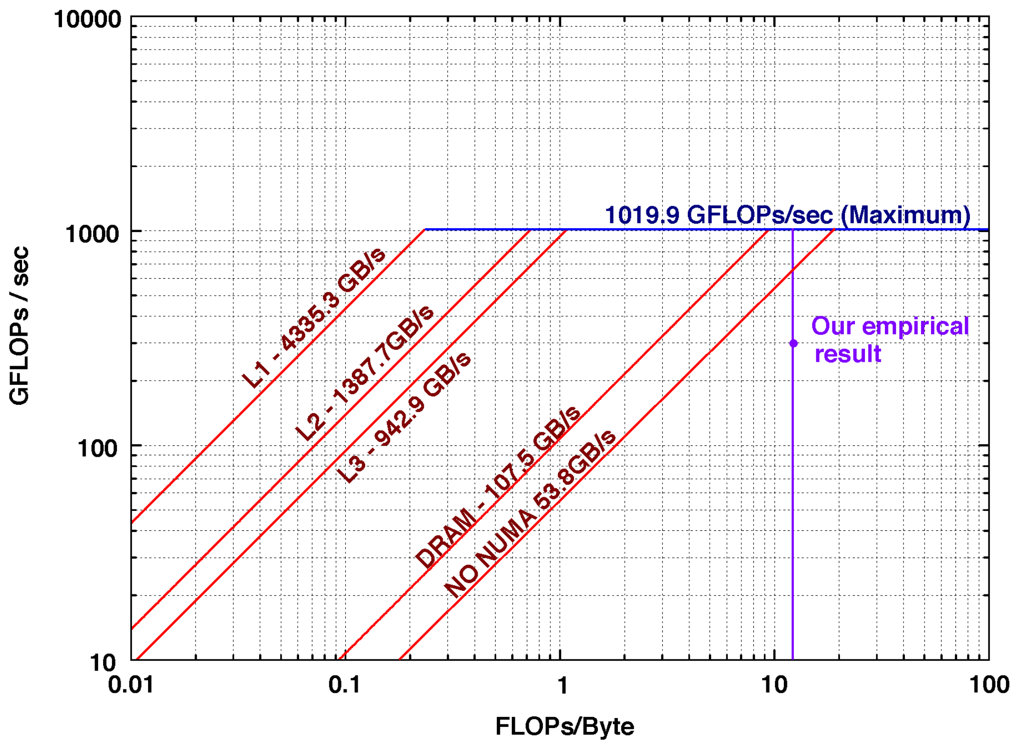
U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory

Arithmetic Intensity & Roofline Model: where we lie in terms of absolute performance

UCLA & NERSC

- Arithmetic intensity is the ratio of computational speed (FLOP's) to data movement (Bytes)
- Nystrom loop runs at 300 GFLOP's and 23 GB/s of DRAM bandwidth giving 13 FLOPs/Byte
- Use Roofline Toolkit (https://bitbucket.org/berkeleylab/cs-roofline-toolkit) to generate bounding "ceilings" on performance



Use Intel's Software Development Emulator Toolkit (SDE) to record FLOP's and Intel's VTune Amplifier to collect data movement when running on 32 cores of a Cori Phase I node

Conclusions

- New class of data clustering methods for segmentation of large datasets with graph based structure
- Fast and accurate method combines ideas from classical nonlinear PDE-based image segmentation with linear algebra methods
- The algorithms can solve semi-supervised and unsupervised graph cut optimization problems
- We give results for hyperspectral video segmentation
- OpenMP parallelization with algorithmic optimization yields nearly ideal scaling on Haswell and KNL