A High-Throughput Solver for Marginalized Graph Kernels on GPU

Yu-Hang Tang, Oguz Selvitopi, Doru Popovici, Aydin Buluc
Computational Research Division
Lawrence Berkeley National Laboratory
PART 1
Marginalized graph kernel for learning on graphs

PART 2
GPU-accelerated high throughput solver

SUMMARY
PART 1

Marginalized graph kernel for learning on graphs
Scientific machine learning (SciML) is a core component of artificial intelligence (AI) and a computational technology that can be trained, with scientific data, to augment or automate human skills. Across the Department of Energy (DOE), scientific machine learning (SciML) has the potential to transform science and energy research.

DOE Basic Research Needs Workshop for Scientific Machine Learning: Core Technologies for Artificial Intelligence 2019
The successes of scientific machine learning have concentrated on select forms of data

**Literature information extraction**
- Text data
- Linear sequence
- Weston et al. 2019

**Climate analytics**
- Grid data
- Real values
- Kurth et al. 2018

**Fluid Mechanics**
- Mesh data
- Real values
- Raissi et al., 2020
Many scientific data and representations are beyond mere images or linear sequences.

- **Molecules**
- **Road network**
- **Social network**
- **Fragmentation tree**

- Variable in size
- Non-sequential
- Mixed continuous/discrete DOFs

Existing solutions often resort to pixelating the data.

Some icons made by Freepik from www.flaticon.com
Graph is a powerful format for scientific data, but machine learning on graphs takes extra effort.

- A graph is a structure that contains objects of pairwise relationships.

- Most existing ML methods work on feature vectors, images, and sequences only.

![Graph example]
A kernel is a function that implicitly transforms raw data into high-dimensional feature vectors via a feature map, and then returns an inner product between the feature vectors. Must be positive-definite.

A kernel is useful for factor out knowledge on data representation from downstream algorithms, and exploit infinite dimensionality and nonlinear feature spaces.

Kernels are used in Support vector machine (SVM), Gaussian process regression (GPR), Kernel principal component analysis (kPCA), etc.

Kernel method in machine learning: what, why, and how
Many ML algorithms have a kernelized counterpart

- SVM: $w^T \phi(x_n) + b$
- Ridge Regression: $w^T \phi(x_n)$
- PCA: $\text{eig}(\Phi^T \Phi)$
- Clustering: Euclidean distance
- Random Projection: Project on random vectors
- SVM with kernel: $\sum a_n t_n k(x, x_n) + b$
- Ridge Regression with kernel: $k(x)^T (K + \lambda I)^{-1} t$
- PCA with kernel: $\text{eig}[k(x_i, x_j)]$
- Clustering with kernel: Kernel-induced distance
- Random Projection with kernel: Project on random samples
Graph kernels are kernels that act on graphs.

<table>
<thead>
<tr>
<th>Graph kernels</th>
<th>Method</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1 × G2 inner product</td>
<td><strong>Histogram</strong></td>
<td>Limited-size subgraphs [Ahmed et al., 2015]</td>
</tr>
<tr>
<td></td>
<td>Statistical moments</td>
<td>[Debnath et al., 1991]</td>
</tr>
<tr>
<td></td>
<td>Random walk</td>
<td>Exponential [Vishwanathan, 2010]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Geometric [Vishwanathan, 2010]</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Marginalized</strong> [Kashima et al., 2003]</td>
</tr>
<tr>
<td></td>
<td>Misc.</td>
<td>[Shervashidze et al. 2011]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Morris et al. 2017]</td>
</tr>
<tr>
<td></td>
<td>Weisfeiler-Lehman</td>
<td>Shortest-path [Borgwardt and Kriegel, 2005]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Spanning tree [Ramon and Gärtner, 2003]</td>
</tr>
</tbody>
</table>
The marginalized graph kernel can seamlessly handle diverse types of graphs

- Definition: the inner product between two graphs is the statistical average of the inner product of simultaneous random walk paths on the two graphs.

**Step 1**
Define random walks

\[ P = D^{-1} \cdot A \]

- \( P \): transition matrix
- \( D \): degree matrix
- \( A \): adjacency matrix

Use edge weight to set transition probability

Sample paths

Compare
The marginalized graph kernel can seamlessly handle diverse types of graphs

- **Definition**: the inner product between two graphs is the statistical average of the inner product of simultaneous random walk paths on the two graphs.

---

**Step 2**
Averaging path similarities

Path similarity defined as product of base kernel evaluations

\[
\kappa_v(\bullet, \bullet) \cdot \kappa_e(\longrightarrow) \cdot \kappa_v(\bullet, \bullet)
\]

\(\cdot\): base kernel for nodes
\(\cdot\): base kernel for edges

---

**Graph inner product/similarity**

**Edge-wise comparison**

**Node-wise comparison**

**Probability**

Wider adoption of the marginalized graph kernel was hindered due to practical challenges

\[ K(G, G') = \sum_{l=1}^{\infty} \sum_{h} \sum_{h'} p_s(h_1)p_s'(h'_1)K_v(v_{h_1}, v'_{h'_1}) \prod_{i=2}^{l} p_t(h_i| h_{i-1})p_q(h_i) \prod_{j=2}^{l} p_t'(h'_j| h'_{j-1})p_q'(h'_j) \prod_{k=2}^{l} K_e(e_{h_{k-1}h_k}, e'_{h'_{k-1}h'_k})K_v(v_{h_k}, v'_{h'_k}) \]

- Cost of computation could be high
  - Direct summation is intractable

- Efficient training involving composite base kernels \( K_v, K_e \) is non-trivial
  - Analytic derivative of the kernel is difficult to derive and implement
Linear algebra reformulation simplifies computations and reveals opportunities for optimization

\[
K(G, G') = \sum_{l=1}^{\infty} \sum_{h} \sum_{h'} p_s(h_1)p'_s(h'_1) K_v(v_{h_1}, v'_{h'_1}) \prod_{i=2}^{l} p_t(h_i|h_{i-1}) p_q(h_i) \prod_{j=2}^{l} p'_t(h'_j|h'_{j-1}) p'_q(h'_j) \prod_{k=2}^{l} K_e(e_{h_{k-1}h_k} e'_{h'_{k-1}h'_k}) K_v(v_{h_k}, v'_{h'_k})
\]

- According to Kashima & Tsuda, the above computation can be simplified into

\[
K(G, G') = \sum_{h} \sum_{h'} s(h_1, h'_1) R_\infty (h_1, h'_1)
\]

where

\[
s(h_1, h'_1) = p_s(h_1)p'_s(h'_1)
\]

\[
R_\infty (h_1, h'_1) = r_1(h_1, h'_1) + \sum_{i,j} t(i, j, h_1, h'_1) R_\infty (h_1, h'_1)
\]

with

\[
t(i, j, h_1, h'_1) = p_t(i|h_1)p'_t(j|h'_1) K_v(v_i, v'_j) K_e(e_{ih_1}, e'_{ih'_1})
\]

- We showed that the formulation is equivalent to the following tensor product linear system:

\[
K(G, G') = p_\times \cdot R_\infty
\]

where \(R_\infty\) can be solved from

\[
[D_\times V_\times^{-1} - A_\times \otimes E_\times] R_\infty = D_\times q_\times,
\]

\[
p_{\times ij} = p_s(i)p'_s(j), q_{\times ij} = p_q(i)p'_q(j)
\]

\[
\text{diag}(D_\times)_{ij} = \text{deg}(v_i) \text{deg}(v'_j)
\]

\[
\text{diag}(V_\times)_{ij} = K_v(v_i, v'_j)
\]

\[
A_{\times ijk} = w_{ij}w'_{kl}
\]

\[
E_{\times ijk} = K_e(e_{ij}, e'_{kl})
\]

multi-index: \(ij: \text{element at } i \cdot n' + j \quad ijk: \text{element at } (ij, kl)\)
Linear algebra reformulation simplifies computations and reveals opportunities for optimization

\[ K(G, G') = \sum_{i=1}^{\infty} \sum_{h} \sum_{h'} p_{s}(h_{i}) p_{s}'(h'_{i}) K_{v}(v_{h_{i}}, v'_{h'_{i}}) \prod_{i=2}^{l} p_{t}(h_{i}|h_{i-1}) p_{q}(h_{i}) \prod_{j=2}^{l} p'_{t}(h'_{j}|h'_{j-1}) p'_{q}(h'_{j}) \prod_{k=2}^{l} K_{e}(e_{h_{k-1}h_{k}}, e'_{h'_{k-1}h'_{k}}) K_{v}(v_{h_{k}}, v'_{h'_{k}}) \]

- The marginalized graph kernel in linear algebra form represents a modified graph Laplacian

\[ K(G, G') = p_{x}^{T} \left( D \times V^{-1} - A \odot E \right)^{-1} D \times q_{x} \]
Linear algebra reformulation simplifies derivation of analytic derivatives

• The gradient of the marginalized graph kernel is crucial for efficient training
• It can be derived using matrix calculus:

\[ K(G, G') = p_x^T \left[ D_x V_x^{-1} - A_x \odot E_x \right]^{-1} D_x q_x \]

Denote

\[ Y = D_x V_x^{-1} - A_x \odot E_x \]

Then

\[ \frac{\partial K}{\partial \theta} = \text{tr} \left[ \frac{\partial K}{\partial Y} \cdot \frac{\partial Y}{\partial \theta} \right] = \left( Y^{-1} p_x \right)^T \frac{\partial Y}{\partial \theta} \left( Y^{-1} D_x q_x \right) \]

Differentiation w.r.t. other hyperparameters can be derived similarly.
Marginalized graph kernel has found successful applications in a variety of ML tasks

- Prediction of molecular atomization energy
  - nodes = atoms, edges = interatomic interactions
  - Jump probabilities proportional to edge weights, which decay with interatomic distance
    \[ w_{ij} = \left(1 - \frac{r_{ij}}{r_c}\right)^n \]
  - Kronecker delta kernel on nodes labeled with chemical elements
    \[ \kappa_v(v_1, v_2) = \begin{cases} 1, & \text{if } v_1 = v_2 \\ h, & \text{otherwise} \end{cases} \]
  - Gaussian kernel on edges labeled by interatomic distance
    \[ \kappa_e(l_1, l_2) = \exp\left[-\frac{1}{2} \frac{(l_1-l_2)^2}{\sigma^2}\right] \]

https://doi.org/10.1063/1.5078640
Marginalized graph kernel has found successful applications in a variety of ML tasks

- Quality assurance on noisy chromatography data

Marginalized graph kernel has found successful applications in a variety of ML tasks

- Protein function prediction

Part 2

GPU-accelerated high throughput solver for marginalized graph kernel
The marginalized graph kernel equation can be efficiently solved using conjugate gradient.

- The conjugate gradient algorithm can be used to **iteratively** solve the marginalized graph kernel equation:
  \[ K(G, G') = \mathbf{p}^T \left( \mathbf{D} \times \mathbf{V}^{-1} \times \mathbf{A} \times \mathbf{E} \right)^{-1} \mathbf{D} \times \mathbf{q} \]

  - \( \mathbf{V} \) and \( \mathbf{E} \) are not necessarily real matrices
  - \( \kappa \) can be complex functions

```python
1 function CG4GK(d,d',V,V',A,A',E,E', q,q')
2     M ← diag \left( (d \otimes d') \odot (V \otimes V')^{-1} \right)
3     x ← 0
4     r ← (d \otimes d') \cdot (q \otimes q')
5     z ← V \otimes V'
6     p ← z
7     ρ ← r^T z
8     repeat
9         a ← (d \otimes d') \odot (V \otimes V')^{-1} \cdot p
10        +(A \otimes A') \odot (E \otimes E') \cdot p
11         α ← ρ/(p^T a)
12         x ← x + α p
13     until r^T r < ε
14     return x
```
Naïve CG on precomputed matrices can only handle small graphs

• Due to the tensor product structure of the linear system, memory usage grows in quartic order

To compute similarity between a pair of 1000-node graphs, a system of $1000000 \times 1000000$ (4TB) is involved.
Naïve CG on precomputed matrices is also memory-bound on GPUs

- NVIDIA Volta GPU requires more than 16 FLOPS per byte (64 FLOPS per float) arithmetic intensity to achieve peak performance
On-the-fly Kronecker matrix-vector multiplication (XMV) can overcome storage and memory bandwidth difficulties

On-the-fly Kronecker matrix-vector multiplication (OTF XMV)

- Regenerates the product linear system on the fly by streaming 8-by-8 submatrices (tiles).
- Tiles staged in shared memory.
- Trade FLOPS for GB/s, but asymptotic arithmetic complexity stays the same.

```
8 repeat
9 \quad a \leftarrow (d \otimes d') \odot (v^v v'^v)^{-1} \cdot p
10 \quad + (A \otimes A') \odot (E \otimes E') \cdot p
11 \quad \alpha \leftarrow \rho / (p^T a)
12 \quad x \leftarrow x + \alpha p
```

OTF XMV achieves much higher FLOPS on dense graphs

- Microbenchmark on V100 with a dot product base kernel
A 2-level hierarchical sparse matrix format ensures efficient memory usage

2-Level sparsity exploitation

- Outer level: retain only non-empty tiles
- Inner level: use bitmap + compact storage format

- **Packing** into compact format: performed on CPU as a preprocessing step
- **Unpacking** for OTF XMV: performed in parallel on GPU using bit magic + warp intrinsics

### Diagram

- **EMPTY TILE DISCARDED**
- **NON-EMPTY TILE COMPRESSED**
- **DENSE STORAGE**

### Bit Map

<table>
<thead>
<tr>
<th>B</th>
<th>D</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>M</td>
<td>Q</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>R</td>
</tr>
<tr>
<td>K</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Dense Storage

```plaintext
0x0303324AE4122041
```

### 64-bit integer nzmask

```plaintext
0b1000001000000100010010000010011101010010010011001100000011000000
```
A 2-level hierarchical sparse matrix format ensures efficient memory usage

- Heuristics for dynamic code path selection:
  - If both tiles contain more than a certain number of non-zero elements, treat them as dense matrices.
  - Otherwise, compute only the non-zeros.

2-Level sparsity exploitation

- Outer level: retain only non-empty tiles
- Inner level: use bitmap + compact storage format

![Diagram of 2-level sparsity exploitation](image)
Specialized graph reordering algorithm improves efficiency of OTF XMV and 2-level sparse format

Partition-based graph reordering (PBR)

- Reduces # of non-empty sparse tiles
- Improves density of non-empty tiles
- Cost easily amortized by repeated pairwise graph kernel computations.
The On-the-Fly GPU Solver Achieves Four Orders of Magnitude Speedup Over Existing Packages

- GraKeL: Cython, multi-threading
- GraphKernels: Python, no parallelization
Marginalized graph kernel has found successful applications in a variety of ML tasks

- Prediction of molecular atomization energy

**Tang & de Jong, J Chem Phys, 2019**
Prediction of atomization energy using graph kernel and active learning
[https://doi.org/10.1063/1.5078640](https://doi.org/10.1063/1.5078640)
Marginalized graph kernel enables active learning of atomization energy in orders of magnitude less time than NN

- QM7: 7165 small organic molecules consisting of H, C, N, O, S, up to 23 atoms
  - From scratch training time: N = 1000: 10 s training, 0.018 s/sample predicting, N = 2000: 40 s training, 0.034 s/sample predicting

MAE: Mean Average Error
RMSE: Root-Mean Square Error
KRR: Kernel Ridge Regression
NN: Neural Network
GRAPE, SOAP, Coulomb, BoB: fingerprint algorithms
SUMMARY
Summary & Acknowledgement

Graphs are useful data structures for representing scientific datasets.

The marginalized graph kernel is a very generic tool for machine learning on graphs.

Marginalized graph kernel can be computed very efficiently on GPUs

• LBNL LDRD Project “Active Learning of Ab Initio Force Fields with Applications to Large-Scale Simulations of Materials and Biophysical Systems”

• Also supported in part by the Applied Mathematics program of the DOE Office of Advanced Scientific Computing Research under Contract No. DE-AC02-05CH11231, and in part by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

Thank You!

pip install graphdot


Manuscript in preparation: GraphDot: A GPU-Accelerated Python Package for Graph-Based Machine Learning.