HTML5 を用いた動的グラフ可視化システムの WebCL による高速化

松林 達史　　星出高秀

日本電信電話株式会社 NTT サイバーソリューション研究所

はじめに

グラフ可視化とは，グラフ $G(V, E)$ として表現する事ができる関係性データを，低次元空間に埋め込むことによって視援を行う技法であり，特に情報工学分野において広く利用されている．グラフ可視化手法では特に力学モデルをベースとした手法が用いられ，N 体力学で用いられている手法の適用が可能となっている．例えば，Tree-code を用いた近似手法や，GPU 等を用いた並列処理による高速化手法などが提案されてきた．

グラフ可視化における問題の一つは，最適なパラメータや収束条件がデータに強く依存し，ユーザーは経験則によりパラメーターの推測を行う必要性が強いという点である．結果として，大規模データの解析においてはパラメータ推定に多くの時間を費やしてしまう．それ故，対話的な処理が可能な，動的大規模グラフ可視化手法が必要となってきている．特にブラウザベースの解析ツールアプリでは，従来よりも JavaScript や flash を利用したグラフ可視化ツールが開発されてきたが，高々 100 ノード程度の処理能力にとどまり，数千ノードを超える大規模なグラフ可視化は非常に困難であった．しかしながら近年では，HTML5 の canvas 機能により描画処理能力は飛躍的に向上し，さらに標準 API の拡充（例えば，WebGL や WebWorkers など）により，ブラウザベースによるグラフ可視化を高速に行うことが可能となった．

そこで我々は，HTML5 の最新機能を用いることにより，対話的かつ高速な動的グラフ可視化手法を開発した．HTML5 の canvas 機能を用いて，動的グラフ可視化における対話的なパラメータ設定を可能とした．また，WebCL を通じてクライアント PC の GPU を利用することにより，JavaScript による処理に対して 200 倍を超える高速化を実現し，数万ノード規模の動的グラフ可視化処理を可能とした．

HTML5 を用いた動的グラフ可視化手法

WebCL による GPU 実装

WebCL とは，OpenCL ベースの HTML5 API であり，JavaScript から処理命令を OpenCL 側に送り，GPU を直接汎用計算に利用することが可能となる．しかしながら，WebCL は標準仕様ではなく，Nokia と Samsung から API が提供されているにとどまっている．Nokia の仕様では firefox のアドオンを利用し OpenCL を直接利用する仕様であり，一方 Samsung の提供する API では，Mac の Web-kit を利用して OpenCL ベースの言語を用いた仕様で，HTML5 の canvas より利用が可能である．

なお，本研究では，Samsung が提供する WebCL の API を用いて GPU 実装を行い，アルゴリズムは比較のため，一般的な $O(|V|^2)$ のアルゴリズムを用いた．図 1(左) は実装コード例である．GPU デバイスの共有メモリを利用し，さらに多次元配列データの 1 次元化を行い，データ転送の効率化を行なっている．

実装速度評価

本研究では，ウェブブラウザには，MacOS の safari 上での JavaScript と WebCL，および Windows7 の Chrome(ver.17) 上での JavaScript を用いて速度評価を行った．ハードウェアは，MacOS には Mac Book Pro (Core2 Duo 2.4GHz + Geforce 9600M(理論性能 120Gflops)) と，Windows には Corei 5(3.6GHz) のデスクトップ PC を用いた．サンプルデータには，$|V| = 62 - 36,458$ の，17 のグラフデータを用いてベンチマークを行った．これらのグラフデータはスパースデータであるゆえ，特に大規模なものは $|V|^2 \gg |E|$ が成り立つ．

横軸をグラフノード数 $|V|$，縦軸を 1 秒あたりの演算回数 (frames per second) で描画したものが，図 1(右) である．ただしここでは演算速度は描画 (canvas の処理時間) にかかる時間を省略している．この図より，数百ノード規模では GPU デバイスとの通信に負荷がかかり，GPU による並列化の恩恵は受けられないということが分かる．しかしながら，1000 ノードを超えると GPU による並列処理の効果が現れ，5,835 ノードは 49.1 倍，14,845 ノードは 109 倍，36,458 ノードでは 200 倍を超える高速化を実現した．なお，safari ブラウザでは，ブラウザがクラッシュを起してたため 1 万ノード以上でのベンチマークの取得を省略している．この結果より，グラフ可視化の最適処理を 1000 回の繰り返し演算と仮定した場合，提案手法では十数分ほどで 3 万ノードを超える大規模なグラフ可視化の計算が可能であることが分かった．なお，今回利用した GPU はモバイル用で，デスクトップ用のハイエンド GPU を用いれば理論的にはさらに 10 倍を超えるの高速化が可能であり，10 万ノード規模の描画処理が可能となる．

対話的グラフ可視化ツール

図 2 は，HTML5 を用いたグラフ可視化システムのスナップショットである．本システムにおいては，計算処理の途中で，引力係数や，斥力係数や，冷却関数係数の値を対話的かつ動的に変更することが可能である．対話的にパラメータ調整を行う事により，バッチ処理によるパラメータサーベイを行わず，ユーザの興味と目的に即した粒度での情報可視化が可能となる．なお，現在はストリーミングデータに対応し，様々な時系列グラフデータをオンラインで描画処理することが可能である [1].

```
__kernel void calc_repulsive_force(  int maxNodes,
                                     float repC,
                                     __global float* nodePos,
                                     __global float* nodeAcc,
                                     __local float* localPos)
{
 unsigned int gid = get_global_id(0);
 if(gid < maxNodes){
   int gid2 = gid * 2 ;
   unsigned int tid  = get_local_id(0);
   unsigned int localSize = get_local_size(0);
   unsigned int numTiles = ceil( (float)( maxNodes * 1.0f / localSize ));
   float2 myPos = (float2) (nodePos[gid2], nodePos[gid2+1]);
   float2 acc = (float2) (0.0f, 0.0f);
   for(int i = 0; i < numTiles; ++i){
     int idx = i * localSize + tid;
     int idx2 = idx * 2;
     int tid2 = tid * 2;
     localPos[tid2  ] = nodePos[idx2  ];
     localPos[tid2+1] = nodePos[idx2+1];
     unsigned int Block = min(localSize, (maxNodes -  i * localSize) );
     barrier(CLK_LOCAL_MEM_FENCE);
     for(int j = 0; j < Block; ++j) {
       if(idx!=gid){
         int jj = jk2;
         float2 aLocalPos = (float2) (localPos[jj], localPos[jj+1]);
         float2 r = aLocalPos - myPos;
         float distSqr = r.x * r.x + r.y * r.y ;
         float force = repC / (distSqr + 0.00001f);
         acc -= force * r;
       }
     }
     barrier(CLK_LOCAL_MEM_FENCE);
   }
   nodeAcc[gid2  ] = acc.x;
   nodeAcc[gid2+1] = acc.y;
 }
}
```

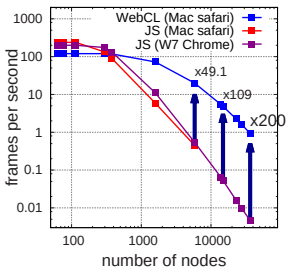

図 1　実装コード例 (左) と，WebCL による高速化の効果 (右).

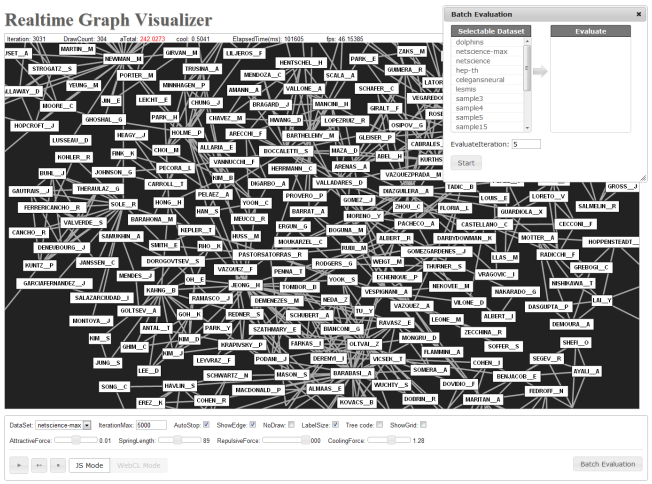

図 2　グラフ可視化ツールのスナップショット．画像はネットワーク科学関連の論文共著者ネットワークの可視化結果.

まとめと今後

本研究では，WebCL を用いることにより，クライアント PC の GPU を利用するシステムを構築し，グラフ可視化計算において JavaScript 実装と比べて 200 倍を超える高速化を実現した．さらに，HTML5 を用いてウェブブラウザベースの実装を行うことによって，オンライン処理による対話的かつ動的なパラメータ調整を可能としたグラフ可視化システムの高速化を実現した．

GPU による汎用計算はここ数年の流行で，NVIDIA や AMD の GPU を利用した研究は盛んに行われて来た．近年ではさらに，GPU を汎用計算に利用する動きがさらに加速しており，Intel IvyBridge CPU では OpenCL からの利用が可能となり，ARM といったモバイル端末用 CPU においても将来的に OpenCL プログラミングが可能となる動きがある．

特にウェブブラウザから並列処理を行う技術に関しては，Intel が ReverTrail プロジェクトを推し進めるなど，HTML5 を用いた技術開発が活発化している．現在 WebCL は標準 API として提供されていないが，今後はウェブブラウザ上からクライアント PC を利用する仕組みは重要な鍵となると考えられる．中でも WebCL は注目するべき技術の 1 つである．我々は今後，HTML5 技術の有効利用を行いつつ，更にオンライン情報解析技術などの開発を目指してゆく予定である．

参考文献

[1] 松林達史、藤村　考、星出 高秀、西田 京介: ”HTML5 を用いたストリーミングデータのグラフ可視化” 信学技報, vol. 112, no. 94, AI2012-1, pp. 1-6, 2012 年 6 月