Vol.2011-ARC-197 No.21
Vol.2011-HPC-132 No.21
2011/11/29

計算機のメモリ階層構造を考慮した高性能ネットワーク解析ライブラリ NETAL

安井 雄一郎[†1,†5] 藤澤 克樹[†1,†5] 佐藤 仁[†2,†5]
鈴村 豊太郎[†2,†3,†5] 後藤 和茂[†4]

様々な分野においてネットワーク解析に対する期待は高まりを見せているものの, 非常に大規模なネットワークを扱うための計算量が課題とされている. そこで我々は, 一般的な計算機環境上での最短路問題と中心性指標に対する, 計算機のメモリ階層構造を考慮した高速計算手法を提案し, NETAL (NETwork Analysis Library) として実装した. NETAL は NUMA アーキテクチャを考慮して, 計算機資源要求の衝突を回避する affinity 設定を行なっている. 実ネットワークに対する数値実験に用いて, 先行研究と比べ最も高速であることを示した. 前処理を必要としない NETAL は, 道路ネットワーク `USA-road-d.USA.gr` に対する全対全最短路長計算を 7.75 日で計算することに成功した. これは Δ-stepping algorithm の 432.4 倍, 9th DIMACS 参照実装 の 228.9 倍の性能に相当する. さらに, GraphCT を用いて 21 日間必要とする `USA-road-d.LKS.gr` に対する *betweenness* 計算は, 我々の実装では複数の中心性指標 *closeness*, *graph*, *stress*, *betweenness* を同時に計算し 1 日で終了する. SSCA#2 を用いた R-MAT グラフに対する *betweenness* 計算に対しても我々の実装は 2.4-3.7 倍の性能を示している.

NETAL: High-Performance Implementation of NETwork Analysis Library Considering Computer Memory Hierarchy

Yuichiro Yasui,[†1,†5] Katsuki Fujisawa,[†1,†5]
Hitoshi Sato,[†2,†5] Toyotaro Suzumura[†2,†3,†5]
and Kazushige Goto[†4]

The use of network analysis has increased in various fields. The large amounts of computation required for dealing with large-scale networks is a major hurdle. We propose an efficient multithreaded computation which considers computer memory hierarchy on general computing environments to solve the shortest paths and the centrality metrics. Our implementation, called NETAL (NETwork Analysis Library), configures the processor core and local memory allocation (affinity), to avoid computational resource request conflicts by considering the difference in distances between processor cores and the RAM within the NUMA architecture of the AMD Opteron 6174. We demonstrated through tests on real-world networks that NETAL is faster than previous implementations. NETAL succeeded in solving the exact *shortest path distance table* for the `USA-road-d.USA.gr` (n =24M, m =58M) without preprocessing in 7.75 days. Numerical results showed that our implementation performance was 432.4 times that of the Δ-stepping algorithm and 228.9 times that of the 9th DIMACS reference solver. Furthermore, while it took GraphCT 21 days to compute the exact *betweenness* of `USA-road-d.LKS.gr`, our implementation computed *multiple centrality metrics* (*closeness*, *graph*, *stress*, and *betweenness*) simultaneously within 1 hour. A performance increase of 2.4-3.7 times compared with R-MAT graph was confirmed for SSCA#2.

1. はじめに

近年, 幅広い分野 (医療, ソーシャルネットワーク, 知識, 生物, 電力網, モデリング, シミュレーション) において, 生じるデータをその関係性から数理的なグラフ・ネットワークとして表現し, その構造を解析する試みが盛んに行われている. 数理分野での理論的な改善と情報技術分野の進歩から, 以前とは比べることのできない規模のネットワークを扱うことができるようになった. しかしながら, 実社会では数千万点を持つ大規模なネットワークが生成されるため, 高性能なグラフ解析ライブラリを整備することは非常に重要である. ネットワーク上の点や枝の重要度判定には中心性指標を利用される機会が多く, 最短路を用いた中心性指標では *closeness*[9], *graph*[10], *stress*[11], *betweenness*[12] などが有名である. 最短路を用いた中心性指標はグラフの接続情報を考慮して, 大域的な情報を考慮することが可能である. 特に, 最短路への寄与率を指標とした媒介中心性 *betweenness* は, 直径の小さな平面性を持たないグラフに対するクラスタリングなどに適用される機会が多い[25]. 最短路を利用した中心性指標は比較的計算量が小さいとされているものの, 厳密もしくはある程度精度を保証

†1 中央大学 / Chuo University
†2 東京工業大学 / Tokyo Institute of Technology
†3 IBM 東京基礎研究所 / IBM Research - Tokyo
†4 マイクロソフト株式会社 / Microsoft Corporation
†5 独立行政法人科学技術振興機構 CREST / JST CREST

Vol.2011-ARC-197 No.21
Vol.2011-HPC-132 No.21
2011/11/29

するためには全対全最短路問題 (all-pairs shortest paths; APSP) と同等の計算量が要求される．*betweenness* に対する効率的な Brendes's algorithm[15),16)] においても, 最短路を求める工程が性能上のボトルネックとされている．

最短路計算に対する既存の実装としては, 9th DIMACS[13)] の参照実装 MLB[3)] や並列アルゴリズム DS[18)] が挙げられる．MLB は高速ではあるが逐次実行のみである．DS は, CRAY XMT などの共有メモリ型多スレッド並列計算機上での並列効率が良いとされているものの, 一般的な Intel や AMD などプロセッサ上でのスレッド並列に対応していない．また, 中心性計算に対する既存の実装として, 高性能なグラフ解析ツール GraphCT[23)], グラフ探索性能ベンチマークソフトウェア SSCA#2[24)] が挙げられる．GraphCT の性能を引き出すためには DS と同様特殊な計算機環境が必要となる．また 32bit 整数型で実装されているため, 扱うことのできる規模が比較的小さい．一方, SSCA#2 は, 前述の共有メモリ型多スレッド並列計算機に加えて, OpenMP による一般的なプロセッサ上でのスレッド並列や, 分散メモリ型クラスタ並列計算機上での MPI 並列に対応しているものの, 対象は人工的に生成したグラフのみである[20),21)]．

そこで我々は, 実社会から生成されたネットワークを対象とし, 最短路と中心性指標に対する一般的な計算機のメモリ階層構造を考慮した効率的なスレッド並列計算手法を提案し, NETAL (NETwork Analysis Library) として実装した[27)]．NETAL は, 与えられた問題の入力 (重みの有無) と出力 (最短路長/全ての最短路の列挙) に応じて子問題に分割しそれらを並列計算する．その際, NUMA アーキテクチャを考慮して, 各スレッドを各プロセッサと各ローカルメモリにバインドする affinity 設定を行いスレッド間の衝突を回避している．

我々は, MLB や DS では数年間必要となる `USA-road-d.USA.gr` に対する全対全最短路長を, NETAL を用いて 7.75 日で計算することに成功した．また, GraphCT を用いて 21 日間必要となる `USA-road-d.LKS.gr` に対する中心性指計算は, NETAL では 1 日で終了する．また, 人工的ネットワークの *betweenness* 計算において, SSCA#2 と比べて, n-BFS で 3.8 倍, n-Dijkstra で 2.4 倍の性能を確認した．GraphCT や SSCA#2 はグラフ内の枝の重みを考慮しない (重みなし) *betweenness* のみの計算に対し, NETAL は 4 種類の中心性 *closeness*, *graph*, *stress*, *betweenness* を同時に計算している．

2. 最短路問題と中心性指標

2.1 最短路問題

最短路問題は始点と終点の個数に関して, BFS, SSSP, MSSP, APSP などに分類することができる．SSSP 問題では, 有向グラフ $G = (V, E)$ と非負枝長 $\ell(e)(e \in E)$, 始点 s を入力とし, s からグラフ内の他の全点 $v \in V$ への最短路を出力する．最短路は, 最短路長 $d_G(s, v)$ と最短路における直前点集合 $\pi(v)$ で表現される．もし, 直前点集合が高々1 点のみを扱う場合 $\pi_s(v) \in V \cup \{\emptyset\}$ には *singlepathSSSP* と, すべての最短路を列挙する場合 $\pi_s(v) = \{u \in V : (u, v) \in E, d_G(s, v) = d_G(s, u) + \ell(u, v)\}$ には *multipathSSSP* と呼ぶことにする．また, 直前点集合を出力しない場合には, *distanceSSSP* と呼ぶことにする．BFS では, 有向グラフ $G = (V, E)$ と始点 s を入力とし, s から $v \in V$ の全点最短距離 (Hop 数) $d_G(s, v)$ と最短路上の直前点集合 $\pi(v)$ を出力する．BFS は枝長を考慮しない SSSP 問題に対応している．SSSP を何度も計算する場合, 複数対全最短路 (*multi-source shortest paths*; MSSP) としてまとめて扱うことができる．MSSP は, 有向グラフ $G = (V, E)$ と非負枝長 $\ell(e)(e \in E)$, 大きさが β である始点集合 $V_S = \{s_0, s_1, ..., s_{\beta-1}\}$ を入力し, 各始点 $s \in V_S$ ごとから他の全点 $v \in V$ へのの最短路 $d_G(s, v)$, $\pi_s(v)$ を出力する．APSP は, 有向グラフ $G = (V, E)$ と非負枝長 $\ell(e)(e \in E)$ を入力とし, 全点間 $(u, v), u, v \in V$ に対する最短路長 $d_G(u, v)$ と, 各点 u を始点とした最短路の直前点集合 $\pi_u(v)$ を出力する．APSP は n 回の BFS, n 回の SSSP, n/β 回の MSSP と同等である．SSSP と同様に, BFS と MSSP, APSP においても, それぞれ, *singlepath*, *multipath*, *distance* に分類できる (表 1).

表 1 最短路問題の出力 (∗∗ には, BFS, SSSP, MSSP, APSP が入る)
Table 1 outputs of various shortest path problems (∗∗ is BFS, SSSP, MSSP, or APSP)

	distance	predecessor list
distance-∗∗	$d_G(s, v)$	—
singlepath-∗∗	$d_G(s, v)$	$\pi_s(v) \in V \cup \{\emptyset\}$
multipath-∗∗	$d_G(s, v)$	$\pi_s(v) = \{u \in V : (u, v) \in E, d_G(s, v) = d_G(s, u) + \ell(u, v)\}$

BFS, SSSP, MSSP に対するアルゴリズムはいずれも, 各点 $v \in V$ に対する $d(v) = \infty$ で初期化された始点からの一時的な距離ラベル $d(v)$ を更新する labeling algorithm である．与えられた始点 $s \in V$ から探索を開始し, 各反復では選択された点 v と接続している点 w の距離ラベルに対する不等式 $d(w) > d(v) + \ell(v, w)$ が成立した場合 $d(w) \leftarrow d(v) + \ell(v, w)$ と更新する．この反復は候補点がなくなるまで繰り返される．探索が終了すると, 一時的な距離ラベル $d(s, v)$ は最短路長 $d_G(s, v)$ となる．labeling algorithm は label setting algorithm もしくは label correcting algorithm に分類されるが, 両者の違いは各反復における点の選択方針である．label setting algorithm では, 最小の $d(v)$ を持つ未確定の点 v が選択され

Vol.2011-ARC-197 No.21
Vol.2011-HPC-132 No.21
2011/11/29

距離ラベルを確定する．そのため, 各点は高々1 度ずつ選択されるため反復回数は最大で点数回となるが, 一般的に並列化は不向きである．一方, label correcting algorithm では, 距離ラベルの小さい順に選択しないためある点が何度も選択される可能性があるが, 点選択コストの削減や並列化が容易である．また, 中心性計算で要求される全ての最短路の列挙や最短路数の出力には不向きである．

2.2 中心性指標

最短路を用いた中心性指標について説明する．まず, 有向グラフ $G = (V, E)$ の各 2 点間 $s, t \in V$ に対して, (s, t)-最短路長を $d_G(s, t)$, (s, t)-最短路数を σ_{st}, v を通る (s, t)-最短路数 σ_{st} とする．ここで, $s = t$ であれば $\sigma_{st} = 1$, $\sigma_{st}(v) = 0$, $v \in \{s, t\}$ となる．最短路を利用した中心性 *closeness* (C_C), *graph* (C_G), *stress* (C_S), *betweenness* (C_B) は表 2 のように定義される．入力のグラフの各枝に重み $\ell(e) \in E$ が与えられる場合, 重み付中心性と定義する．点に対する中心性指標を扱うが, 枝に対しても同様に定義ができる[16]．

表 2 最短路を用いた中心性指標
Table 2 Centrality metrics using shortest paths

closeness[9]	$C_C(v) = \frac{1}{\sum_{t \in V} d_G(v,t)}$	*graph*[10]	$C_G(v) = \frac{1}{\max_{t \in V} d_G(v,t)}$
stress[11]	$C_S(v) = \sum_{s \neq v \neq t \in V} \sigma_{st}(v)$	*betweenness*[12]	$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$

Brandes's algorithm Brandes による *betweenness* C_B に対するアルゴリズム[15),16)]の説明を行う．このアルゴリズムでは, 各点に対して 1 度ずつ「最短路フェイズ」(2 行目) と「中心性更新フェイズ」(3-7 行目) が要求される．最短路フェイズでは始点から他の全点に対する最短路上の直前点集合 $\pi_s(v)$ と各点の最短路の通過数 $\sigma_{st}(v)$, 最短路長の大きい順に並ぶ点列 $\mathcal{S}$ を出力する．また中心性更新フェイズでは得られた点列順に中心性を計算する．以下に示すアルゴリズムは重みなし C_B に対するもので $O(nm)$ の計算量が与えられている．重み付 C_B では, 最短路フェイズを `multipathSSSP` とし, binary heap 付 Dijkstra's algorithm を用いて $O(nm \log n)$ の計算量が要求される．Bader らは, この計算量を削減するランダムサンプリング手法を提案している[19]．この手法では各点 1 度ずつの反復 (1 行目) をランダムに選出したいくつかの点のみ計算し, 得られた指標を $C'_B(v) \leftarrow \frac{|V|}{|V'|} \cdot C_B(v), \forall v \in V$ とすることによって, 近似的な C'_B を得ることができる．

Input: directed graph $G = (V, E)$
Output: $C_B(v), \forall v \in V$ (initialize to 0)
1: **for** $s \in V$ **do**
2: $\sigma, \pi_s, \mathcal{S} \leftarrow$ `multipathBFS`(G, s)
3: **while** $\mathcal{S} \neq \emptyset$ **do**
4: pop $w \leftarrow \mathcal{S}$
5: **for** $v \in \pi_s(w)$ **do** $\delta_B(v) \leftarrow \delta_B(v) + \frac{\sigma(s,v)}{\sigma(s,w)} \cdot (1 + \delta_B(w))$
6: **if** $w \neq s$ **then** $C_B(w) \leftarrow C_B(w) + \delta_B(w)$
7: **end while**
8: **end for**

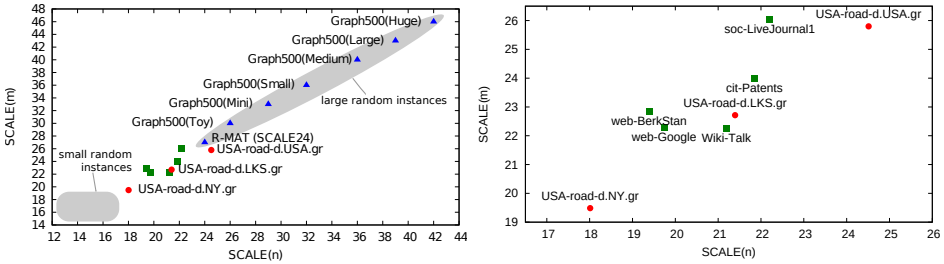
複数中心性指標の同時計算 Brandes's algorithm は必要となる最短路問題と同等の計算量を要求する．各反復で求めた最短路は反復内でしか利用できないため, NETAL は得られた最短路から複数の中心性指標を同時に計算し再利用性を高めている．指標毎に示すグラフ特性は異なるため, 同時に複数の中心性が得られることはネットワーク解析において有用である．計算量が同等の Brandes's algorithm の拡張を用いて, C_C, C_G, C_S, C_B を同時に扱う．

Input: directed graph $G = (V, E)$
Output: $C_C(v)$, $C_G(v)$, $C_S(v)$, $C_B(v)$, $\forall v \in V$ (initialize to 0)
1: **for** $s \in V$ **parallel do**
2: $d_G, \sigma, \pi_s, \mathcal{S} \leftarrow$ `multipathBFS`(G, s)
3: $C_C(s) \leftarrow \frac{1}{\sum_{t \in V} d_G(s,t)}$; $C_G(s) \leftarrow \frac{1}{\max_{t \in V} d_G(s,t)}$
4: **while** $\mathcal{S} \neq \emptyset$ **do**
5: pop $w \leftarrow \mathcal{S}$
6: **for** $v \in \pi_s(w)$ **do** $\delta_S(v) \leftarrow (1 + \delta_S(w))$; $\delta_B(v) \leftarrow \delta_B(v) + \frac{\sigma(s,v)}{\sigma(s,w)} \cdot (1 + \delta_B(w))$
7: **if** $w \neq s$ **then** $C_S(w) \leftarrow C_S(w) + \sigma(s, w) \cdot \delta_S(w)$; $C_B(w) \leftarrow C_B(w) + \delta_B(w)$
8: **end while**
9: **end for**

3. 最短路計算に対する既存研究とその問題点

実ネットワークに対する中心性指標計算においてボトルネックとなる最短路計算に関する既存研究の問題点を, 想定される規模と期待される性能に着目して列挙する．9th DIMACS[13)] や SNAP Project[14)] で公開されている実ネットワークの規模は, 図 1 (b) が示すように点数 $n = [2^{18}, 2^{25}]$ である．先行研究は主に 3 種類 (1) 索引付けによる高速化, (2) 小規模な APSP に対する並列計算, (3) 大規模な BFS や SSSP に対する並列計算 に分類される． (1) 9th DIMACS では点数 $n = 24M$, 枝数 $m = 58M$ の全米道路ネットワークを対

Vol.2011-ARC-197 No.21
Vol.2011-HPC-132 No.21
2011/11/29



(a) real-world/random graph instance size　　(b) real-world instance size

図 1 (a) 実社会ネットワークとランダムグラフの大きさ, (b) 実社会ネットワークの大きさ, 横軸は点数 (対数) $\log n$, 縦軸は枝数 (対数) $\log m$.

Fig. 1 (a) scale of real-world/random graph, (b) scale of real-world graph. The x-axis is the number of nodes on a log scale n, and y-axis is the number of arcs on a log scale m.

象に 2 点間最短路計算の高速化について活発な議論が行われた. 最も高速な Transit Node Routing は, 数時間程度の前処理による索引付けを用いて, 始終点ペアあたり 4.9 μ 秒と非常に高速に答えることができる[5]. しかしながら最短路長のみを出力するため, 中心性計算への利用は容易ではない. また索引付けには緯度経度などの地理情報を必要となるために, 性能がグラフ特性に影響を受けるとされるが, 道路ネットワーク以外の適用事例は示されておらず性能は明らかではない. (2) 比較的小さな規模 ($n = [2^{12}, 2^{17}]$, 図 1 (a)) の APSP に対しては, GPGPU を用いた実装[6),7)] や, MSSP に対する Dijkstra's algorithm を拡張した *multi-source label-correcting algorithm* (MSLC)[8)] が提案されている. 入力の規模を限定した高速化を行なっており, 想定より大きな規模に対しては, メモリ要求量の増大のために実行不能となる, 性能効率が極端に低下する, といった状況が予想される. (3) 1 台の計算機上では配置できない規模 ($n = [2^{26}, 2^{42}]$, 図 1 (a)) に対しては, BFS の並列アルゴリズムである *level-synchronized parallel BFS algorithm* (LS-BFS)[17], SSSP に対する並列アルゴリズムである Δ-*stepping algorithm* (DS)[18)] が提案されている. 性能を引き出すためには, 計算機環境は分散メモリ型クラスタ並列計算機や共有メモリ型多スレッド計算機などの特殊な計算機が必要となる. 次数が大きく直径が小さい大規模なグラフを想定しており, 対象のネットワークでは性能を引き出すことは困難である. また, Super Computing 2010 では Graph500 List[26)] と呼ばれるグラフ探索性能による計算機性能評価が提案された. Kronecker product で生成される scale-free 性を持つ Kronecker graph を対象として, BFS の *traversed edges per second* (TEPS) 値によりランキングが決定される.

表 3 は, 各枝に $[0, 1024]$ の乱数を割り当てた 256×256 の格子型グラフ `Square.16.0.gr` (点数 $n = 65536$, 枝数 $m = 261120$, 最大枝長 $C = 1024$) に対する, Xeon X5460 上でのアルゴリズム毎の逐次性能にまとめたものである. 表中の Complexity は各問題あたりの計算量, CPU time は APSP に要する実行時間 (秒) を示している. APSP を分割して計算する際, 繰り返し計算する BFS, SSSP, MSSP に対するアルゴリズムに全体の性能が依存するため, 逐次性能だけでなく並列性能も考慮したアルゴリズム選択が非常に重要である. DIKF の結果や文献[2)] が示すように, 必ずしも計算量から期待される性能が得られるとは限らないため, 数値実験による評価が必要となる.

表 3 Xeon X5460 における APSP (`Square.16.0.gr`) に対する各アルゴリズムの逐次性能 (CPU time (秒))
Table 3 Sequential performance (CPU time in seconds) of APSP (`Square.16.0.gr`) computation attained by each algorithm on Xeon X5460

implementation	algorithm	complexity	CPU time
BFS (breadth-first search)			
BFS[13)]	naïve BFS	$O(m)$	530.99
LS-BFS[17)]	level-synchronized parallel BFS	$O(m)$	117.10
Label Setting Algorithm (Dijkstra's algorithm) for SSSP			
DIKQ[2)]	naïve Dijkstra's algorithm	$O(n^2)$	12610.28
DIKH[2)]	k-heap ($k = 4$)	$O(m \log_k n)$	961.89
DIKB[2)]	Dial's algorithm	$O(m + nC)$	771.02
DIKBD[2)]	double buckets $\Delta = \lceil C/2^{11} \rceil$	$O(m + n(\Delta + C/\Delta))$	875.69
DIKF[2)]	Fibonacci heaps	$O(m + n \log n)$	2310.56
MLB[3)]	multi-level buckets	$O(m + n \log C)$	869.23
Label Correcting Algorithm for SSSP			
BFM[2)]	naïve Bellman-Ford-Moore algorithm	$O(nm)$	3050.17
DS[18)]	Δ-stepping algorithm	$O(dn)$	1244.08
Label Correcting Algorithm for MSSP ($\beta = 128$ sources)			
MSLC[8)]	multi-source label correcting algorithm	$O(\beta m \log n)$	118.02

4. 計算機のメモリ階層構造を考慮した効率的なスレッド並列計算手法

NETAL は, 最短路と中心性に対する, 一般的な計算機のメモリ階層構造を考慮した効率的なスレッド並列計算を行う. 中心となるのは APSP に対する 3 種類の戦略 n-BFS, n-Dijkstra, n/β-MSLC で, n-BFS と n-Dijkstra は点数 n 個に分割した multipathBFS と multipathSSSP を, n/β-MSLC は n/β 個に分割した β 始点の distanceMSSP を, そ

Vol.2011-ARC-197 No.21
Vol.2011-HPC-132 No.21
2011/11/29

れぞれ並列計算する. 加えて, n-BFS, n-Dijkstra を用いた高速な中心性計算を提供する. 並列計算の際には, 我々の先行研究 2-HEAP[4] の成果に加え, NUMA アーキテクチャを考慮して, 各スレッドを各プロセッサと各ローカルメモリに固定する affinity 設定を行いスレッド間の衝突の回避とメモリ参照の改善を行った.

4.1 計算機資源要求を考慮した高速化

2-HEAP は計算機資源要求を考慮した高速化を行った SSSP に対する binary heap 付 Dijkstra's algorithm である. 計算機資源要求の衝突による性能低下の割合を性能評価に用いて, 並列実行性能を重視した高速化を行った. 近年のプロセッサのメモリ階層構造は複雑化し各コア間の距離は一定ではないため, 並列実行時のスレッドの競合による性能低下が問題である. 2-way Xeon X5460 (図 2) においても, 各コアの RAM までの距離は一定であるものの, コア間は距離が大きい順に, "**different processors**", "(same processor) **different L2 caches**", "(same processor) **same L2 cache**" となる組合せが存在する (図 3). Xeon X5460 ではメモリバスを共有しているため, 逐次実行時 (**sequential**) と

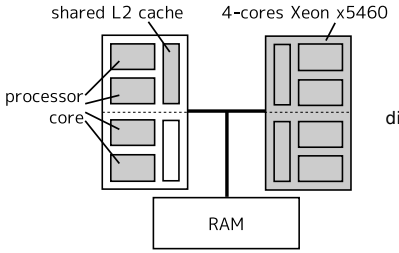


図 2 2-way Xeon X5460
Fig. 2 2-way Xeon X5460

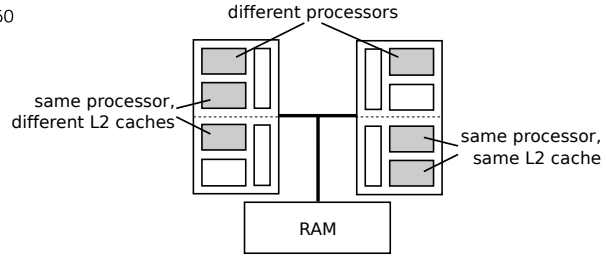


図 3 2-way Xeon X5460 の 2 コア
Fig. 3 two-core combination of 2-way Xeon X5460

比べ, different processors による 2 並列時にはコアと RAM 間の帯域幅が相対的に半分となる. また, different L2 caches は different processors に比べてプロセッサ内部での帯域幅が相対的に半分となる. さらに same L2 cache は, different L2 caches に比べて共有した L2 cache のうち半分の領域, 帯域しか使用できない. このように, プロセッサ間の距離が近づき共有する計算機資源の割合が大きくなるに従い, 資源要求の衝突により性能低下が予想される. つまり計算機資源要求の衝突による性能低下を観察することによって, 階層構造上のボトルネック箇所を特定を行うことができる (表 4). この解析方法では各コアの組

表 4 計算機資源競合によるメモリ階層構造におけるボトルネック解析
Table 4 Bottleneck analysis of the memory hierarchy by considering the rate of performance reduction due to simultaneous execution

bottleneck	different processors	different L2 caches	same L2 cache
processor ↔ RAM bandwidth	down	down	down
processor inside bandwidth	-	down	down
L2 cache sharing	-	-	down
Arithmetic performance	-	-	-

合せ上で同一の逐次実行プログラムを同時に実行し性能計測を行うが, 一方のプロセスによってキャッシュ上に配置されたデータをもう一方のプロセスが再利用することはないことに注意されたい. 表 5 は, 表 3 で比較したアルゴリズムの同時実行時の性能であり, 我々の 2-HEAP は高速 (実行時間が短い) かつ高効率 (資源要求競合が小さい) であることを示している. また, bucket 系アルゴリズムは, heap 系アルゴリズムと比べて計算機資源に対する要求が厳しく, 並列数の増加に従い性能低下の増大が予想される. 表 6 は道路ネットワークに対する 2-HEAP のスレッド並列計算性能をまとめたものである. 2-HEAP は, 4 並列時には "same L2 cache" を回避し理想的な台数効果を示すものの, 8 並列時では "same L2 cache" による計算機資源の競合により効率の低下が確認される. これは表 5 の結果と一致し, 資源要求解析により並列性能特性の見積りが可能であることを示している.

表 5 2-way Xeon X5460 の 2 コア同時実行による計算機資源競合解析 (CPU time (秒), 性能比率 (%))
Table 5 Computational results (CPU time in seconds and performance ratio %) for simultaneous tests using two processor cores on a 2-way Xeon X5460

	sequential	different processors	different L2 caches	same L2 cache
2-HEAP[4]	5.34 (± 0.00%)	5.44 (-1.93%)	5.62 (- 5.05%)	6.63 (-18.94%)
DIKH($k = 4$)	7.23 (± 0.00%)	7.26 (-0.41%)	7.59 (- 4.74%)	8.79 (-17.75%)
DIKF	15.95 (± 0.00%)	16.09 (-0.87%)	16.56 (- 3.68%)	18.17 (-12.22%)
DIKB	4.38 (± 0.00%)	4.54 (-3.52%)	5.01 (-12.58%)	6.38 (-31.35%)
DIKBD	4.65 (± 0.00%)	4.88 (-4.71%)	5.25 (-11.43%)	6.64 (-29.97%)
MLB	5.69 (± 0.00%)	5.85 (-2.74%)	6.17 (- 7.78%)	7.73 (-26.39%)
DS	11.74 (± 0.00%)	12.06 (-2.66%)	12.55 (- 6.41%)	16.49 (-28.76%)

4.2 NUMA アーキテクチャを考慮したデータ参照の改善

最新のプロセッサアーキテクチャは, Xeon X5460 (図 2) のような UMA (Uniform Memory Access) から NUMA (Non-Uniform Memory Access) に移行している. NUMA アーキテクチャである 12-core Opteron 6174 (図 4) では, 各コアごとに L2 cache が, 6 コア

Vol.2011-ARC-197 No.21
Vol.2011-HPC-132 No.21
2011/11/29

表 6 2-way Xeon X5460 における SSSP に対する並列性能 (CPU time (秒), 性能向上率, メモリ要求量)
Table 6 Parallel performance (CPU time in seconds, speedup ratio, and memory requirement) of SSSP on a 2-way Xeon X5460

	`USA-road-d.NY.gr` ($n = 264K, m = 734K$)		`USA-road-d.USA.gr` ($n = 24M, m = 58M$)	
	CPU time (speedup)	MB	CPU time (speedup)	GB
2-HEAP (sequential)	35.69 (× 1.00)	10.70	5300.8 (× 1.00)	0.90
2-HEAP (2 threads)	18.02 (× 1.98)	14.80	2734.4 (× 1.94)	1.27
2-HEAP (4 threads)	8.94 (× 3.99)	22.99	1451.3 (× 3.65)	2.00
2-HEAP (8 threads)	4.85 (× 7.36)	39.38	1031.7 (× 5.14)	3.46
MLB (sequential)	41.53	25.32	5873.0	2.17

ごとに共有の L3 cache がそれぞれ配置している．プロセッサに直接接続されている RAM (local memory) と, 他のプロセッサ下に配置している RAM (remote memory) が存在するため, 各プロセッサコアと各 RAM の距離が異なる．remote memory へのアクセスは HT (Hyper Transport) で接続されたプロセッサ経由であるため干渉による性能低下が予想される．このように NUMA アーキテクチャでは, コアとメモリの距離を考慮した適切な計算機資源の割振が重要であるものの, 現在の OS には効率よくコアの割当を行う機能はなく実装者に任せられている． そこで n-BFS, n-Dijkstra, n/β-MSLC では, 2-HEAP の成果に

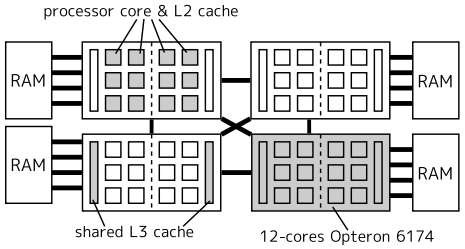


図 4 NUMA アーキテクチャ 4-way Opteron 6174
Fig. 4 NUMA architecture 4-way Opteron 6174

加えて, 適切な各スレッドをプロセッサと RAM を固定する affinity 設定を行い干渉の回避を行った．ここで, コアと RAM を関連付ける affinity 設定を「**バインドしたコア数 × バインド数**」と表すことにする．この設定は, バインド毎にグラフデータ (もしくは複製) を配置し, バインドされたコアから参照するように固定する．図 5 は 48×1, 図 6 は 6×8 の affinity 設定を表している．コアとスレッドの固定には `set_sched_affinity` 関数を用いた．

表 7, 8, 9 は, `USA-road-d.NY.gr` ($n = 264K$, $m = 734K$) に対する Affinity 毎の APSP

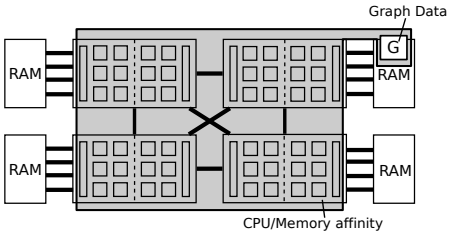


図 5 4-way Opteron 6174 における 48×1 affinity 設定 (worst)
Fig. 5 48×1 affinity (worst) on a 4-way Opteron 6174

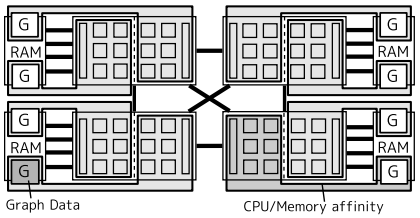


図 6 4-way Opteron 6174 に おける 6×8 affinity 設定 (best)
Fig. 6 6×8 affinity (best) on a 4-way Opteron 6174

の実行時間 (1 スレッドからの性能向上率), TEPS 値, 使用メモリ量を, n-BFS, n-Dijkstra, n/β-MSLC($\beta = 32$) 毎にまとめたものである．いずれの実装においても, local memory を共有したコアをバインドする 6×8 の affinity 設定が最も性能が高い．Opteron 6174 では L3 cache を共有した 6 コア毎にメモリバスが接続しているためである．一方, remote memory に頻繁にアクセスする affinity 設定 (48×1) は, 設定を行わない場合と比べ性能が悪化してしまう．このように, 適切な affinity 設定によって得られる性能向上だけでなく, 誤った設定による性能低下にも注意しなければならない．6×8 の affinity 設定を行った 48 スレッド並列計算において, n-BFS は 46.21 倍, n-Dijkstra は 43.6 倍, n/β-MSLC($\beta = 32$) は 36.7 倍 の台数効果が得られた． また, 実行時間の分散をまとめた図 7 により, 適切な設定は実行時間の安定化に効果があることが示された．適切な設定 (best) は高速で実行時間も安定し, 誤った設定 (worst) は低速で実行時間も不安定となる．

4.3 MSLC における始点数の適正値と優先キューに用いる優先度

n/β-MSLC は主に始点数 β の適正値と優先キューの優先度 $k(v)(v \in V)$ によって性能が依存する．n/β-MSLC では始点数分 β の距離ラベル用のデータ領域が必要となるため, 適正値が大きい場合メモリ要求量によって最高の性能を出すことができない可能性が考えられる．始点数と優先キューの優先度による n/β-MSLC の性能をまとめた表 10 は, 我々の実装では適正値 $\beta = 32$ と小さく, 64bit 整数型で実装しているためより規模の大きなグラフへの対応が可能であることを示している．一方, 先行研究の柳澤の実装は, 適正値が $\beta = 128$ と

Vol.2011-ARC-197 No.21
Vol.2011-HPC-132 No.21
2011/11/29

表 7 APSP (`USA-road-d.NY.gr`) に対する n-BFS の並列性能 (CPU time (秒), TEPS, メモリ要求量 (MB))
Table 7 Parallel performance (CPU time in seconds, TEPS ratio, and memory space in Mbytes) of n-BFS for APSP (`USA-road-d.NY.gr`) on a 4-way Opteron 6714

	affinity	CPU time (speedup ratio)	MTEPS	MB
sequential	default	9449.8 (× 1.0)	20.5	53.1
12 threads	default	821.4 (×11.5)	236.2	292.2
	12 × 1	810.4 (×11.7)	239.4	292.2
	{1, 2} × 8	648.1 (×14.6)	299.3	398.8
24 threads	default	426.1 (×22.2)	387.8	553.0
	24 × 1	500.2 (×18.9)	387.8	553.0
	3 × 8	353.1 (×26.8)	549.4	659.6
48 threads	default	346.2 (×27.3)	560.3	1074.6
	48 × 1	343.1 (×27.5)	565.4	1074.6
	6 × 8	204.5 (×46.2)	948.6	1181.2

表 8 APSP (`USA-road-d.NY.gr`) に対する n-Dijkstra の並列性能 (CPU time (秒), TEPS, メモリ要求量 (MB))
Table 8 Parallel performance (CPU time in seconds, TEPS ratio, and memory space in Mbytes) of n-Dijkstra for APSP (`USA-road-d.NY.gr`) on a 4-way Opteron 6714

	affinity	CPU time (speedup ratio)	MTEPS	MB
sequential	default	17915.8 (× 1.0)	10.8	53.1
12 threads	default	1804.7 (× 9.9)	107.5	292.2
	12 × 1	1613.8 (×11.1)	120.2	292.2
	{1, 2} × 8	1444.5 (×12.4)	134.3	398.8
24 threads	default	875.1 (×20.5)	221.7	553.0
	24 × 1	910.6 (×19.7)	213.0	553.0
	3 × 8	776.7 (×23.1)	249.8	659.6
48 threads	default	517.2 (×34.6)	375.0	553.0
	48 × 1	549.6 (×32.6)	353.0	1074.6
	6 × 8	411.2 (×43.6)	471.7	1181.2

大きくメモリ要求量に対する要求が厳しいこと, 32bit 浮動小数点数を採用しているため精度不足が生じる可能性があること, などが挙げられ大規模なグラフ上での実行が困難である. また, 優先キューに用いる優先度は, 先行研究と同様, 各点 v における始点 $s \in V_S$ ごとの距離ラベル $d(s, v)$ の最小値を採用する PQ_min $k(v) = \min\{d(s, v) | s \in V_S\}$ が最も良い.

表 9 APSP (`USA-road-d.NY.gr`) に対する n/β-MSLC($\beta = 32$) の並列性能 (CPU time (秒), TEPS, メモリ要求量 (MB))
Table 9 Parallel performance (CPU time in seconds, TEPS ratio, and memory space in Mbytes) of n/β-MSLC($\beta = 32$) for APSP (`USA-road-d.NY.gr`) on a 4-way Opteron 6714

	affinity	CPU time (speedup ratio)	MTEPS	MB
sequential	default	1584.4 (× 1.0)	122.4	109.8
12 threads	default	159.2 (×10.0)	1218.5	1036.6
	12 × 1	170.7 (× 9.2)	1136.6	1036.6
	{1, 2} × 8	135.6 (×11.7)	1430.9	1143.2
24 threads	default	80.2 (×19.8)	2418.6	2047.6
	24 × 1	94.1 (×16.8)	2060.8	2047.6
	3 × 8	73.6 (×21.5)	2634.0	2154.2
48 threads	default	51.1 (×31.0)	3799.9	4069.7
	48 × 1	51.0 (×31.1)	3805.6	4069.7
	6 × 8	44.4 (×35.7)	4372.5	4176.3

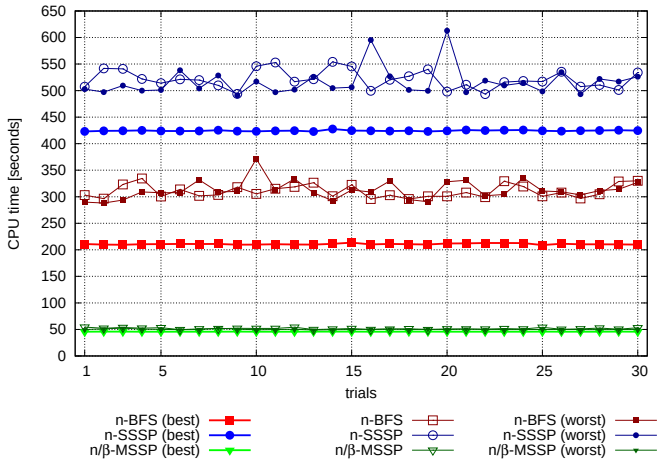


図 7 4-way Opteron 6174 における APSP (`USA-road-d.NY.gr`) の実行時間
Fig. 7 CPU time for APSP (`USA-road-d.NY.gr`) on a 4-way Opteron 6174

5. 数値実験

我々の 3 種類の実装 n-BFS, n-Dijkstra, n/β-MSLC を用いた, APSP と中心性計算に

表 10 4-way Opteron 6174 における `USA-road-d.NY.gr` の APSP の n/β-MSLC マルチスレッド計算における探索戦略と β の適正値

Table 10 Parallel performance (CPU time in seconds) of priority queue strategy for the n/β-MSLC multithreaded computation of APSP (`USA-road-d.NY.gr`) and the appropriate value of the source parameter β for a 4-way Opteron 6174

strategy	priority queue key $k(v)$ $(v \in V)$	$\beta = 16$	$\beta = 32$	$\beta = 48$
PQ_min	$k(v) = \min\{d(s,v) \mid s \in V_S\}$	53.4 s	**44.4** s	44.6 s
PQ_max	$k(v) = \max\{d(s,v) \mid d(s,v) \neq \infty, s \in V_S\}$	59.2 s	56.4 s	60.9 s
PQ_avg	$k(v) = \sum_{s \in V_S, d(s,v) \neq \infty} d(s,v)$	801.9 s	1241.1 s	1477.1 s

対する数値実験による性能評価を行う．我々の実装はいずれも C 言語で 64bit 整数型を用いて実装している．数値実験に利用した計算機は 4-way Opteron 6174 2.2 GHz (12 cores × 4), 256 GB の RAM で, 搭載 OS は Fedora 15 64bit, C コンパイラは GCC 4.6.0 である．APSP に対する実験結果 (表 12) と中心性計算に対する実験結果 (表 13) における * が付与したデータは推定値である．実験に用いたグラフインスタンスは, 9th DIMACS[13] の対象である道路ネットワーク, SNAP Project[14] で公開されている様々な実ネットワーク, SSCA#2 で生成される人工ネットワーク[22] である．`web-BerkStan`, `web-Google`, `Wiki-Talk`, `cit-Patents` に対する厳密な直径 $diam(G) = \max_{s,t \in V} d_G(s,t)$ (枝長は考慮しない) を求めることに成功した．各実験で使用した実装を表 11 にまとめる．

5.1 全対全最短路 (APSP)

直径が大きな道路ネットワークと直径が小さな実ネットワークに対する APSP の結果を表 12 にまとめる．いずれのインスタンスに対しても, 我々の実装は既存の実装と比べて数十倍の性能を示している．`USA-road-d.NY.gr` では MLB と比べて n-Dijkstra は 32.3 倍, n/β-MSLC ($\beta = 32$) では 298.7 倍の性能が確認される．また他の実装では数ヶ月間から数年間の計算量を必要とする, 全米道路ネットワークである `USA-road-d.USA.gr` を n/β-MSLC ($\beta = 16$) に用いて 7.75 日 で, Live-Journal のソーシャルネットワークである `soc-LiveJournal1` を n/β-MSLC ($\beta = 32$) を用いて 2.78 日で, それぞれ厳密な全対全最短路長を求めることに成功した．

5.2 中心性指標

表 13 は, 我々の実装 n-BFS と n-Dijkstra (重み付中心性) を用いた複数中心性指標 *closeness*, *graph*, *stress*, *betweenness* の計算性能と, 解析ライブラリである GraphCT における *betweenness* の計算性能をまとめる．表中の上段では厳密な計算時間と TEPS 値を, 下段では実行時間中の最短路フェイズ (sp) と中心性更新フェイズ (up) が占める割合を示し

表 11 全対全最短路計算と中心性計算に対する実装

Table 11 Implementations for APSP and centrality computation

		algorithm	shortest paths centrality	computation
APSP	NETAL (this paper)			
	n-BFS	breadth-first search (BFS)	multipathBFS $\times n$	parallel
	n-Dijkstra	Dijkstra with binary heap	multipathSSSP $\times n$	parallel
	n/β-MSLC	multi-source label correcting	distanceMSSP $\times n/\beta$	parallel
	other implementations			
	LS-BFS[17]	level-synchronized parallel BFS	singlepathBFS $\times n$	sequential
	MLB[3]	Dijkstra with multi-level buckets	singlepathSSSP $\times n$	sequential
	DS[18]	Δ-stepping algorithm	distanceSSSP $\times n$	sequential
Centrality	NETAL (this paper)			
	n-BFS	breadth-first search (BFS)	C_C, C_G, C_S, C_B	parallel
	n-Dijkstra	Dijkstra with binary heap	C_C, C_G, C_S, C_B (weighted)	parallel
	other implementations			
	GraphCT[23]	Level-synchronized parallel BFS	C_B	sequential
	SSCA#2[24]	Level-synchronized parallel BFS	C_B	parallel

表 12 4-way Opteron 6174 における APSP に対する数値実験結果 (CPU time, TEPS 値)

Table 12 Computational results (CPU time and TEPS ratio) of APSP on 4-way Opteron 6174

	`USA-road-d.NY.gr`	`USA-road-d.LKS.gr`	`USA-road-d.USA.gr`
	$n = 264K$	$n = 2.76M$	$n = 24.0M$
	$m = 734K$	$m = 6.89M$	$m = 58.3M$
	$diam = 720$	$diam = 4127$	$diam = 8352*$
n-BFS	3.41 min	10.54 hours	* 70 days
n-Dijkstra	6.85 min	15.45 hours	* 99 days
n/β-MSLC ($\beta = 16$)	0.89 min	1.69 hours	7.75 days
n/β-MSLC ($\beta = 32$)	0.74 min	1.43 hours	memory over
LS-BFS	* 55 min	* 126 hours	* 557 days
MLB	* 221 min	* 430 hours	* 1774 days
DS ($\Delta = 0.1$)	* 365 min	* 695 hours	* 3351 days

	`web-BerkStan`	`web-Google`	`Wiki-Talk`	`cit-Patents`	`soc-LiveJournal1`
	$n = 685K$	$n = 876K$	$n = 2.39M$	$n = 3.78M$	$n = 4.85M$
	$m = 7.60M$	$m = 5.12M$	$m = 5.02M$	$m = 16.52M$	$m = 68.99M$
	$diam = 714$	$diam = 51$	$diam = 11$	$diam = 24$	$diam = 18*$
n-BFS	23.10 min	77.35 min	53.74 min	93.38 min	* 7.51 days
n-Dijkstra	44.83 min	108.23 min	76.61 min	188.44 min	* 9.63 days
n/β-MSLC ($\beta = 32$)	9.69 min	33.63 min	37.88 min	121.65 min	2.78 days
LS-BFS	* 621 min	* 1998 min	* 621 min	* 569 min	* 113.1 days
MLB	* 1847 min	* 1946 min	* 1847 min	* 1940 min	* 103.4 days
DS ($\Delta = 10.0$)	* 957 min	* 6904 min	* 3573 min	* 15096 min	* 288.6 days

* estimated value

Vol.2011-ARC-197 No.21
Vol.2011-HPC-132 No.21
2011/11/29

ている．我々の実装 n-BFS や n-Dijkstra は GraphCT に比べていずれのインスタンスに対しても数十倍以上の性能を示している．最も実行時間を必要とする `USA-road-d.LKS.gr` では, GraphCT を用いて 20.6 日間かかる計算が, n-BFS や n-Dijkstra では 1 日で計算が終了する．このとき GraphCT に比べて n-BFS では 25.5 倍, n-Dijkstra では 21.6 倍の性能となる．さらに 表 14 に SSCA#2 との比較をまとめる．同条件での比較のために, SSCA#2 が生成する SCALE24 の R-MAT グラフ ($n = 2^{SCALE}, m = 8n$) を使用し, SSCA#2 がランダムサンプリングに指定した 256 始点を使用した．GraphCT は使用したデータ型 (32bit 整数) の範囲を超えてしまい, エラーで終了してしまった．スレッド並列計算に対応した SSCA#2 と比べて我々の実装は, n-BFS では 3.8 倍, n-Dijkstra では 2.4 倍の性能を示している．

表 13 4-way Opteron 6174 における 9thDIMACS/SNAP に対する中心性計算の結果 (CPU time, TEPS 値, 最短路フェイズ (sp) と更新フェイズ (up) の割合)

Table 13 Computational results (CPU time, TEPS ratio, and the ratio of the shortest-path phase(sp) and update phase(up)) of centrality for 9thDIMACS/SNAP instances on 4-way Opteron 6174

instance	n-BFS (C_C, C_G, C_S, C_B)	n-Dijkstra (weighted C_C, C_G, C_S, C_B)	GraphCT (C_B)
`USA-road-d.NY.gr`	0.11 h (0.474 GTEPS)	0.17 h (0.312 GTEPS)	* 3.66 h
$n = 264K, m = 734K$	sp: 49.50 %, up: 50.07 %	sp: 67.66 %, up: 32.06 %	
`web-BerkStan`	0.66 h (2.208 GTEPS)	1.05 h (1.381 GTEPS)	* 17.99 h
$n = 685K, m = 7.60M$	sp: 60.33 %, up: 39.51 %	sp: 71.83 %, up: 28.07 %	
`web-Google`	2.15 h (0.578 GTEPS)	2.70 h (0.462 GTEPS)	* 52.97 h
$n = 876K, m = 5.12M$	sp: 60.61 %, up: 39.34 %	sp: 68.04 %, up: 31.92 %	
`Wiki-Talk`	2.05 h (1.631 GTEPS)	2.57 h (1.297 GTEPS)	* 22.10 h
$n = 2.39M, m = 5.02M$	sp: 42.15 %, up: 57.75 %	sp: 51.93 %, up: 47.98 %	
`USA-road-d.LKS.gr`	19.35 h (0.273 GTEPS)	22.84 h (0.231 GTEPS)	* 493.77 h
$n = 2.76M, m = 6.89M$	sp: 54.52 %, up: 45.46 %	sp: 69.48 %, up: 30.50 %	
`cit-Patents`	1.87 h (9.243 GTEPS)	2.52 h (6.865 GTEPS)	* 23.61 h
$n = 3.78M, m = 16.52M$	sp: 27.11 %, up: 72.68 %	sp: 39.65 %, up: 60.20 %	

* estimated value

表 14 R-MAT graph (SCALE24) における 中心性計算 (256 始点) の Opteron 6714 並列性能 (CPU time (秒), TEPS 値)

Table 14 Opteron 6714 parallel performance (CPU time in seconds and TEPS ratio) of centrality computation (256-random-sampling) on R-MAT graph (SCALE24).

instance	n-BFS (C_C, C_G, C_S, C_B)	n-Dijkstra (weighted C_C, C_G, C_S, C_B)	GraphCT (C_B)	SSCA#2 (C_B)
R-MAT graph	163.0 seconds	260.5 seconds	error	620.0 seconds
$n = 2^{24}, m = 2^{27}$	210.8 MTEPS	131.9 MTEPS		48.5 MTEPS

6. まとめと今後の課題

現在の計算機環境はコア数の増大とメモリ要求量により, これまでとは比較にならない規模の問題を扱うことが可能になっている．今後の更なる計算機のメモリ階層構造の多階層化・複雑化が予測されており, 理論と実験の両側面を考慮した改善が極めて重要であると考えている．そこで, 基本的な組合せ最適化問題である最短路問題に対して, 一般的な計算機環境である 4-way Opteron6174 上での計算機資源を効率よく使用するスレッド並列を提案し, NETAL (NETwork Analysis Library) ライブラリとして実装した．本実装 NETAL は, 前処理による索引付けを必要としない, 全ての最短路を出力する n-BFS, n-Dijkstra(重みを考慮), 最短路長を計算する n/β-MSLC(重みを考慮) で構成される．スレッド並列計算の際, NUMA アーキテクチャのコアと RAM の距離を考慮した affinity 設定を行い, 計算機資源要求の衝突を回避した．`USA-road-d.NY.gr` ($n = 264M, m = 783K$) に対する全対全最短路問題では, 最短路長を 44.4 秒 (n/β-MSLC), 枝長を考慮しない最短路を 204.5 秒 (n-BFS), 枝長を考慮した最短路を 411.2 秒 (n-Dijkstra) と高速に求め, それぞれ 35.7 倍, 46.2 倍, 43.8 倍の台数効果が得られた．n-Dijkstra と n/β-MSLC は, 9th DIMACS 参照実装と比べて 32.3 倍と 298.7 倍の性能, Δ-stepping と比べて 53.3 倍と 493.24 倍の性能を示している．また, 先行研究では数年間必要とする `USA-road-d.USA.gr` ($n = 24.0M, m = 58.3M$) に対する全対全最短路長を 7.75 日 (始終点あたり 1.167 ナノ秒) で計算することに成功した．さらに, `USA-road-d.LKS.gr` ($n = 2.76M, m = 6.89M$) に対する中心性計算においては, 先行研究 GraphCT (21 日間) に比べ n-BFS で 25.5 倍 (19.4 時間), n-Dijkstra で 21.6 倍 (22.8 時間) の性能を確認した．また SSCA#2 に対し n-BFS で 3.8 倍, n-Dijkstra で 2.4 倍の性能を確認した．さらに我々の実装では複数の中心性 *closeness*, *graph*, *stress*, *betweenness* を同時に扱ってるため, *betweenness* のみを計算する GraphCT や SSCA#2 と比較して適用範囲が広い．成果は, 4-way Opteron6174 以外の NUMA アーキテクチャプロセッサを保有する計算機環境に対しても適用可能である．その際, 同様の affinity 設定による効率的な計算が期待できる．また, 最短路計算や中心性計算以外の独立した子問題を何度も計算することが要求される, 様々なアルゴリズムに対し適用が期待できる．我々は単純なグラフ解析に留まらず, 大規模災害時などの突発的な事態に対する避難経路探索・交通管制システムの構築を目指している．そのようなシステムではある程度頻繁に更新が行われるデータに対して高速に処理することが要求されるため, 更なる高速化が必要となると考える．

謝辞 本研究の一部は科学技術振興機構 (JST) の戦略的創造研究推進事業「CREST」に

Vol.2011-ARC-197 No.21
Vol.2011-HPC-132 No.21
2011/11/29

おける研究領域「ポストペタスケールシステムにおける超大規模グラフ最適化基盤」の支援を受けています.

参 考 文 献

1) E. W. Dijkstra: *A Note on Two Problems in Connexion with Graphs*. Numerische Mathematik, 1:269.271 (1959).
2) B. V. Cherkassky, A. V. Goldberg, T. Radzik: *Shortest paths algorithms: theory and experimental evaluation*. Mathematical programming (1996).
3) A.V. Goldberg: *A simple shortest path algorithm with linear average time*. In Algorithm - ESA 2001, 9th Annual European Symposium, Lecture Notes in Computer Science, 2161 (2001), pp.230-241 (2001).
4) 安井雄一郎, 藤澤克樹, 笹島啓史, 後藤和茂：大規模最短路問題に対するダイクストラ法の高速化, 日本オペレーションズリサーチ学会論文誌, Vol.54, 10-17 (2011).
5) H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes: *In Transit to Constant Time Shortest-Path Queries in Road Networks*. SIAM Workshop on Algorithms Engineering and Experiments (ALENEX '07), pp.46-59 (2007).
6) P.Harish, P.J.Narayanan: *Accelerating Large Graph Algorithms on CPU Using CUDA*, HiPC 2007, LNCS 4873, Springer-Verlag Berlin Heidelberg, pp.284-291, (2007).
7) T.Okuyama, F.Ino, K.Hagihara: *A Task Parallel Algorithm for Computing the Costs of All-Pairs Shortest Paths on CUDA-compatible GPU*, International Symposium on Parallel and Distributed Processing with Applications, IEEE, pp.284-291, (2008).
8) H.Yanagisawa: *A Multi-Source Label Correcting Algorithm for the All-Pairs Shortest Paths Problem*, Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium (2010).
9) G. Sabidussi: *The centrality index of a graph*. Psychometrika, 31: pp.581-603 (1966).
10) P. Hage and F. Harary: *Eccentricity and centrality in networks*. Social Networks, 17: pp.57-63 (1995).
11) A. Shimbel: *Structural parameters of communication networks*. Bulletin of Mathematical Biophysics, 15: pp.501-507 (1953).
12) L.C. Freeman: *A set of measures of centrality based on betweenness*. Sociometry, 40: pp.35-41 (1977).
13) 9th DIMACS Implementation Challenge. `http://www.dis.uniroma1.it/~challenge9/`.
14) SNAP: Stanford Network Analysis Project. `http://snap.stanford.edu/`.
15) U. Brandes, *A Faster Algorithm for Betweenness Centrality*, Journal of Mathematical Sociology 25(2), pp.163-177 (2001).
16) U. Brandes, *On Variants of Shortest-Path Betweenness Centrality and their Generic Computation*, Social Networks 30, 2, pp.136-145 (2008).
17) D.A. Bader and K. Madduri. *Designing multithreaded algorithms for breadth-first search and st-connectivity on the Cray MTA-2*. In Proc. 35th Int'l Conf. on Parallel Processing (ICPP), Columbus, OH, August 2006. IEEE Computer Society (2006).
18) K. Madduri, D.A. Bader, J.W. Berry, and J.R. Crobak, *Parallel Shortest Path Algorithms for Solving Large-Scale Instances*, 9th DIMACS Implementation Challenge – The Shortest Path Problem, DIMACS Center, Rutgers University, Piscataway, NJ, November 13-14 (2006).
19) D.A. Bader, S. Kintali, K. Madduri, and M. Mihail, *Approximating Betweenness Centrality*, The 5th Workshop on Algorithms and Models for the Web-Graph (WAW2007), San Diego, CA, December 11-12 (2007).
20) K. Madduri, D. Ediger, K. Jiang, D. A. Bader, D. C.-Miranda: *A Faster Parallel Algorithm and Efficient Multithreaded Implementations for Evaluating Betweenness Centrality on Massive Datasets*, IPDPS '09 Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing, IEEE Computer Society Washington, DC, USA (2009).
21) D.A. Bader and K. Madduri: *Parallel algorithms for evaluating centrality indices in real-world networks*, in Proc. 35th Int'l Conf. on Parallel Processing (ICPP 2006), (Columbus, OH), pp.539-550, IEEE Computer Society, Aug (2006).
22) D. Chakrabarti, Y. Zhan, and C. Faloutsos: *R-MAT: A recursive model for graph mining*, SIAM Data Mining (2004).
23) D. Ediger, K. Jiang, J. Riedy, D.A. Bader, C. Corley: *Massive Social Network Analysis: Mining Twitter for Social Good*, Parallel Processing, International Conference on, pp.583-593, 39th International Conference on Parallel Processing (2010).
24) D.A. Bader, K. Madduri, J. Gilbert, V. Shah, J. Kepner, T. Meuse, and A. Krishnamurthy: *Designing scalable synthetic compact applications for bench- marking high productivity computing systems*, CTWatch Quarterly, vol. 2, Nov (2006).
25) D.A. Bader and K. Madduri: *SNAP, Small-world Network Analysis and Partitioning: an open-source parallel graph framework for the exploration of large-scale networks*, in Proc. 22nd Int'l Parallel and Distributed Processing Symp. (IPDPS 2008), (Miami, FL), Apr (2008).
26) Graph500.org - The Graph 500 List, `http://www.graph500.org/`.
27) Y.Yasui, K.Fujisawa, K.Goto, N.Kamiyama, and M.Takamatsu: *NETAL:High-Performance Implementation of Network Analysis Library Considering Computer Memory Hierarchy*. Journal of the Operations Research Society of Japan (submitted).