Parallel Computing 59 (2016) 43–59

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

A parallel min-cut algorithm using iteratively reweighted least squares targeting at problems with floating-point edge weights

Yao Zhu*, David F. Gleich

Department of Computer Science, Purdue University, 305 N. University Street, West Lafayette, IN 47907, USA

ARTICLE INFO

Article history:
Received 30 March 2015
Revised 21 November 2015
Accepted 22 February 2016
Available online 3 March 2016

Keywords:
Undirected graphs
s–t min-cut
Iteratively reweighted least squares
Laplacian systems
Parallel linear system solvers

ABSTRACT

We present a parallel algorithm for the undirected *s–t* min-cut problem with floating-point valued edge weights. Our overarching algorithm uses an iteratively reweighted least squares framework. Specifically, this algorithm generates a sequence of Laplacian linear systems, which are solved in parallel. The iterative nature of our algorithm enables us to trade off solution quality for execution time, which is distinguished from those purely combinatorial algorithms that only produce solutions at optimum. We also propose a novel two-level rounding procedure that helps to enhance the quality of the approximate min-cut solution output by our algorithm. Our overall implementation, including the rounding procedure, demonstrates significant speed improvement over a state-of-the-art serial solver, where it could be up to 200 times faster on commodity platforms.

1. Introduction

We consider the undirected *s–t* min-cut problem. Our goal is a practical, scalable, parallel algorithm for problems with hundreds of millions or billions of edges and with floating point weights on the edges. Additionally, we expect there to be a sequence of such *s–t* min-cut computations where the difference between successive problems is small. The motivation for such problems arises from a few recent applications including the FlowImprove method to improve a graph partition [1] and the GraphCut method to segment high-resolution images and MRI scans [2,3]. Both of these applications are limited by the speed of current *s–t* min-cut solvers, and the fact that most of them cannot handle problems with floating point edge weights. We seek to accelerate such *s–t* min-cut computations, especially on floating point valued instances.

For the undirected *s–t* min-cut problem, we present a Parallel Iteratively Reweighted least squares Min-Cut solver, which we call PIRMCut for convenience. This algorithm draws its inspiration from the recent theoretical work on using Laplacians and electrical flows to solve max-flow/min-cut in undirected graphs [4–6]. However, our exposition and derivation is entirely self-contained. In contrast to traditional combinatorial solvers for this problem, our method produces an approximate min-cut solution, just like many of the recent theory papers [4–6].

There are three essential ingredients to our approach. The first essential ingredient is a variational representation of the ℓ_1-minimization formulation of the undirected *s–t* min-cut (Section 2.1). This representation allows us to use the iteratively reweighted least squares (IRLS) method to generate a sequence of symmetric diagonally dominant linear systems whose

* Corresponding author. Tel.: +1 7654305239.
E-mail addresses: yaozhu@purdue.edu (Y. Zhu), dgleich@purdue.edu (D.F. Gleich).

http://dx.doi.org/10.1016/j.parco.2016.02.003
0167-8191/

solutions converge to an approximate solution (Theorem 2.6). We show that these systems are equivalent to electrical flows computation (Proposition 2.3). We also prove a Cheeger-type inequality that relates an undirected s–t min-cut to a generalized eigenvalue problem (Theorem 2.7). The second essential ingredient is a parallel implementation of the IRLS algorithm using a parallel linear system solver. The third essential ingredient is a two-level rounding procedure that uses information from the electrical flow solution to generate a much smaller s–t min-cut problem suitable for serial s–t min-cut solvers.

The current state-of-the-art combinatorial s–t max-flow/min-cut solvers [7–10] are all based on operating the residual graph. The residual graph and associated algorithms are usually associated with complex data structures and updating procedures. Thus, the operations on it would result in irregular memory access patterns. Moreover, the directed edges of the residual graph come and go frequently during algorithm execution. This dynamically changing structure of the residual graph further exacerbates the irregularity of the s–t min-cut computation if the combinatorial solvers are used. In contrast, PIRMCut reduces the s–t min-cut problem to solving a sequence of Laplacian systems all with the same fixed nonzero structure. Because then only matrix computations are used, such a reduction also gets rid of the need for complex updating procedures. Although it does not necessarily wipe out the irregularity of the application, it does significantly diminishes the degree of irregularity. In fact, it has been demonstrated that irregular applications rich in parallel sparse matrix computations can obtain significant speedups on multithreaded platforms [11]. In a broader sense, the algorithmic paradigm embodied by PIRMCut, i.e., reducing irregular graph applications to more regular matrix computations, could facilitate the adoption of modern high performance computing systems and architectures, especially those specifically designed for irregular applications [12–14].

We have designed and implemented an MPI based implementation of PIRMCut and evaluated its performance on both distributed and shared memory machines using a set of test problems consisting of different kinds of graphs. Our solver, PIRMCut, is 200 times faster (using 32 cores) than a state-of-the-art serial s–t min-cut solver on a test graph with no essential difference in quality. In the experimental results, we also demonstrate the benefit of using warm starts when solving a sequence of related linear systems. We further show the advantage of the proposed two-level rounding procedure over the standard sweep cut in producing better approximate solutions.

At the moment, we do not have a precise and graph-size based runtime bound on PIRMCut. We also acknowledge that, like most numerical solvers, it is only up to δ-accurate. The focus of this paper is investigating and documenting a set of techniques that are principled and could lead to practically fast solutions on real world s–t min-cut problems. We compare our approach with those of others and discuss some further opportunities of our approach in the related work and discussions (Sections 4 and 6).

2. An IRLS algorithm for undirected s–t min-cut

In this section, we describe the derivation of the IRLS algorithm for the undirected s–t min-cut problem. We first introduce our notations. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a weighted, undirected graph. In the derivation that follows, the term *weight* will be reserved to denote the set of weights that result from the IRLS algorithm. Thus, at this point, we wish to refer to the *edge weights* as *capacities* following the terminology in network flow problems. Let $n = |\mathcal{V}|$, and $m = |\mathcal{E}|$. We require for each undirected edge $\{u, v\} \in \mathcal{E}$, its capacity $c(\{u, v\}) > 0$. Let s and t be two distinguished nodes in $\mathcal{G}$, and we call s the *source node* and t the *sink node*. The problem of undirected s–t min-cut is to find a partition of $\mathcal{V} = \mathcal{S} \cup \bar{\mathcal{S}}$ with $s \in \mathcal{S}$ and $t \in \bar{\mathcal{S}}$ such that the cut value

$$\mathrm{cut}(\mathcal{S}, \bar{\mathcal{S}}) = \sum_{\substack{\{u,v\}\in\mathcal{E} \\ u\in\mathcal{S}, v\in\bar{\mathcal{S}}}} c(\{u, v\})$$

is minimized. In the interest of solving the undirected s–t min-cut problem, we assume $\mathcal{G}$ to be connected. We call the subgraph of $\mathcal{G}$ induced on $\mathcal{V}\backslash\{s, t\}$ the *non-terminal graph*, and denote it by $\widetilde{\mathcal{G}} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}})$. We call the edges incident to s or t the *terminal edges*, and denote them by $\mathcal{E}^{\mathcal{T}}$.

The undirected s–t min-cut problem can be formulated as an ℓ_1-minimization problem. Let $\mathbf{B} \in \{-1, 0, 1\}^{m\times n}$ be the edge-node incidence matrix corresponding to an arbitrary orientation of $\mathcal{G}$'s edges, and $\mathbf{C}$ be the $m \times m$ diagonal matrix with $c(\{u, v\})$ on the main diagonal. Further let $\mathbf{f} = [1\ 0]^{\mathrm{T}}$, and $\Phi^{\mathrm{T}} = [\mathbf{e}_s\ \mathbf{e}_t]$ where $\mathbf{e}_s$ ($\mathbf{e}_t$) is the sth (tth) standard basis, then the undirected s–t min-cut problem is

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & ||\mathbf{CBx}||_1 \\ \text{subject to} \quad & \Phi\mathbf{x} = \mathbf{f}, \quad \mathbf{x} \in [0, 1]^n. \end{aligned} \tag{1}$$

Note that in (1) we adopt the constraint $\mathbf{x} \in [0, 1]^n$ instead of the integral constraint $\mathbf{x} \in \{0, 1\}^n$. This change is justified because once the st min-cut problem is converted into a linear program in standard form, the matrix $\mathbf{B}$ appears in the constraints. The incidence matrix $\mathbf{B}$ is a standard example of totally unimodular [15]. Thus, such a relaxation does not change the set of integral optimal solutions.

2.1. The IRLS algorithm

The IRLS algorithm for solving the ℓ_1-minimization problem (1) is motivated by the variational representation of the ℓ_1 norm

$$\|\mathbf{z}\|_1 = \min_{\mathbf{w}\geq 0} \tfrac{1}{2}\sum_i \left(\frac{\mathbf{z}_i^2}{\mathbf{w}_i} + \mathbf{w}_i\right).$$

Using this variational representation, we can rewrite the ℓ_1-minimization problem (1) as the following joint minimization problem:

$$\underset{\mathbf{x},\mathbf{w}\geq 0}{\text{minimize}}\ H(\mathbf{x},\mathbf{w}) = \frac{1}{2}\sum_i \left(\frac{(\mathbf{C}\mathbf{B}\mathbf{x})_i^2}{\mathbf{w}_i} + \mathbf{w}_i\right) \tag{2}$$

$$\text{subject to } \Phi\mathbf{x} = \mathbf{f}, \quad \mathbf{x}\in[0,1]^n \tag{3}$$

The IRLS algorithm for (1) can be derived by applying alternating minimization to (2) [16]. Given an IRLS iterate $\mathbf{x}^{(l-1)}$ satisfying constraint (3), the next IRLS iterate $\mathbf{x}^{(l)}$ is defined according to the following two alternating steps:

- *Step 1:* Compute the *reweighting vector* $\mathbf{w}^{(l)}$, of which each component is defined by

$$\mathbf{w}_i^{(l)} = \sqrt{(\mathbf{C}\mathbf{B}\mathbf{x}^{(l-1)})_i^2 + \epsilon^2} \tag{4}$$

  where $\epsilon > 0$ is a smoothing parameter that guards against the case of $(\mathbf{C}\mathbf{B}\mathbf{x}^{(l-1)})_i = 0$.
- *Step 2:* Let $\mathbf{W}^{(l)} = \text{diag}(\mathbf{w}^{(l)})$, update $\mathbf{x}^{(l)}$ by solving the weighted least squares (WLS) problem

$$\underset{\mathbf{x}}{\text{minimize}}\quad \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{C}(\mathbf{W}^{(l)})^{-1}\mathbf{C}\mathbf{B}\mathbf{x} \quad \text{subject to} \quad \Phi\mathbf{x} = \mathbf{f}. \tag{5}$$

We prove that the sequence of iterates from the IRLS algorithm is well-defined, that is, each iterate $\mathbf{x}^{(l)}$ exists and satisfies $\mathbf{x}^{(l)} \in [0,1]^n$. Solving (5) leads to solving the saddle point problem

$$\begin{bmatrix} \mathbf{L}^{(l)} & \Phi^{\mathrm{T}} \\ \Phi & \mathbf{O} \end{bmatrix}\begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{f} \end{bmatrix} \tag{6}$$

where $\boldsymbol{\lambda}$ is the Lagrange multiplier corresponding to the constraint $\Phi\mathbf{x} = \mathbf{f}$, and $\mathbf{L}^{(l)}$ is defined as follows:

$$\mathbf{L}^{(l)} = \mathbf{B}^{\mathrm{T}}\mathbf{C}(\mathbf{W}^{(l)})^{-1}\mathbf{C}\mathbf{B} \tag{7}$$

We note that $\mathbf{L}^{(l)}$ is a *reweighted Laplacian* of $\mathcal{G}$, where the edge capacities $\mathbf{C}$ are reweighted by the diagonal matrix $(\mathbf{W}^{(l)})^{-1}$. For a recent survey on graph Laplacians, we refer to [17]. Regarding the nonsingularity of the linear system (6), we have the following proposition.

Proposition 2.1. *A sufficient and necessary condition for the linear system (6) to be nonsingular is that $\mathcal{G}$ is connected.*

Proof. When $\mathcal{G}$ is connected, $\ker(\mathbf{L}^{(l)}) = \text{span}\{\mathbf{e}\}$, where $\mathbf{e}$ is the all-one vector. From the definition of Φ, we know $\mathbf{e}\notin\ker(\Phi)$, i.e., $\ker(\mathbf{L}^{(l)})\cap\ker(\Phi) = \{0\}$. With Φ being full rank, we apply Theorem 3.2 from [18] to prove the proposition. □

We now prove that the solution $\mathbf{x}^{(l)}$ to (6) automatically satisfies the interval constraint.

Proposition 2.2. *If $\mathcal{G}$ is connected, then for each IRLS update we have $\mathbf{x}^{(l)} \in [0,1]^n$.*

Proof. According to proposition 2.1, the linear system (6) is nonsingular when $\mathcal{G}$ is connected. We apply the null space method [18] to solve it. Let $\mathbf{Z}$ be the matrix from deleting the s, t columns of the identity matrix $\mathbf{I}_n$, then $\mathbf{Z}$ is a null basis of Φ. Because $\Phi\mathbf{e}_s = \mathbf{f}$, using $\mathbf{Z}$ and $\mathbf{e}_s$ we reduce (6) to the following linear system:

$$(\mathbf{Z}^{\mathrm{T}}\mathbf{L}^{(l)}\mathbf{Z})\mathbf{v} = -\mathbf{Z}^{\mathrm{T}}\mathbf{L}^{(l)}\mathbf{e}_s \tag{8}$$

where $\mathbf{v}$ is the vector by deleting the s and t components from $\mathbf{x}$. Note $\mathbf{b}^{(l)} = -\mathbf{Z}^{\mathrm{T}}\mathbf{L}^{(l)}\mathbf{e}_s \geq 0$ because $\mathbf{L}^{(l)}$ is a Laplacian. We call $\widetilde{\mathbf{L}}^{(l)} = \mathbf{Z}^{\mathrm{T}}\mathbf{L}^{(l)}\mathbf{Z}$ the *reduced Laplacian*, and $(\widetilde{\mathbf{L}}^{(l)})^{-1} \geq 0$ because it's a Stieltjes matrix (see Corollary 3.24 of [19]). Thus $\mathbf{v}^{(l)} = (\widetilde{\mathbf{L}}^{(l)})^{-1}\mathbf{b}^{(l)} \geq 0$. Also note that $\widetilde{\mathbf{L}}^{(l)}\mathbf{e} \geq \mathbf{b}^{(l)}$, thus $\widetilde{\mathbf{L}}^{(l)}(\mathbf{e}-\mathbf{v}^{(l)}) = \widetilde{\mathbf{L}}^{(l)}\mathbf{e} - \mathbf{b}^{(l)} \geq 0$. Using $(\widetilde{\mathbf{L}}^{(l)})^{-1} \geq 0$ again, we have $\mathbf{e}-\mathbf{v}^{(l)} \geq 0$. In summary, we have $0 \leq \mathbf{v}^{(l)} \leq \mathbf{e}$. □

In applying the IRLS algorithm to solve the undirected s–t min-cut problem, we start with an initial vector $\mathbf{x}^{(0)}$ satisfying (3). We take $\mathbf{x}^{(0)}$ to be the solution to (5) with $\mathbf{W}^{(0)} = \mathbf{C}$. Then we generate the IRLS iterates $\mathbf{x}^{(l)}$ for $l = 1,\ldots,T$ using (4) and (5) alternatingly. Finally we apply a rounding procedure on $\mathbf{x}^{(T)}$ to get a binary cut indicator vector. We discuss the details of the rounding procedure in Section 3.4. The majority of the work the IRLS algorithm does is in solving a sequence of reduced Laplacian systems for $l = 1,\ldots,T$. It is already known that undirected max-flow/min-cut can be approximated by solving a sequence of electrical flow problems [4]. We make the connection between the IRLS and the electrical flow explicit by

Proposition 2.3. Because of this connection between IRLS and the electrical flow, we also call $\mathbf{x}^{(l)}$ the *voltage vector* in the following.

Proposition 2.3. *The WLS (5) solves an electrical flow problem. Its flow value is* $(\mathbf{x}^{(l)})^{\mathrm{T}}\mathbf{L}^{(l)}\mathbf{x}^{(l)}$.

Proof. Let $\mathbf{x}^{(l)}$ be the solution to (5) and $\mathbf{z}^{(l)} = \mathbf{C}(\mathbf{W}^{(l)})^{-1}\mathbf{C}\mathbf{B}\mathbf{x}^{(l)}$. From $\frac{1}{2}(\mathbf{x}^{(l)})^{\mathrm{T}}\mathbf{L}^{(l)}\mathbf{x}^{(l)} = \frac{1}{2}(\mathbf{z}^{(l)})^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{W}^{(l)}\mathbf{C}^{-1}\mathbf{z}^{(l)}$, $\mathbf{z}^{(l)}$ minimizes the energy function $E(\mathbf{z}) = \frac{1}{2}\mathbf{z}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{W}^{(l)}\mathbf{C}^{-1}\mathbf{z}$. We now prove that $\mathbf{z}^{(l)}$ satisfies the flow conservation constraint,

$$\begin{aligned}\mathbf{B}^{\mathrm{T}}\mathbf{z}^{(l)} &= \mathbf{B}^{\mathrm{T}}\mathbf{C}(\mathbf{W}^{(l)})^{-1}\mathbf{C}\mathbf{B}\mathbf{x}^{(l)} = \mathbf{L}^{(l)}\mathbf{x}^{(l)}\\ &= -\Phi^{\mathrm{T}}\boldsymbol{\lambda}\end{aligned}$$

where the last equality is by the first equation in (6). From $\Phi^{\mathrm{T}} = [\mathbf{e}_s\ \mathbf{e}_t]$, we have $(\mathbf{B}^{\mathrm{T}}\mathbf{z}^{(l)})_u = 0$ when $u \neq s, t$. Also note that $\mathbf{e}^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{z}^{(l)} = 0$. Thus $\mathbf{z}^{(l)}$ is an electrical flow. From $\mathbf{x}_s^{(l)} = 1$ and $\mathbf{x}_t^{(l)} = 0$, the flow value of $\mathbf{z}^{(l)}$ is given by

$$\begin{aligned}\mu(\mathbf{z}^{(l)}) &= (\mathbf{x}^{(l)})^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{z}^{(l)} = (\mathbf{x}^{(l)})^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{C}(\mathbf{W}^{(l)})^{-1}\mathbf{C}\mathbf{B}\mathbf{x}^{(l)}\\ &= (\mathbf{x}^{(l)})^{\mathrm{T}}\mathbf{L}^{(l)}\mathbf{x}^{(l)}.\end{aligned}$$

□

2.2. Convergence of the IRLS algorithm

We now apply a general proof of convergence from Beck [16] to our case. Because of the smoothing parameter ϵ introduced in defining the reweighting vector $\mathbf{w}^{(l)}$ as in (4), the IRLS algorithm is effectively minimizing a smoothed version of the ℓ_1-minimization problem (1) defined by

$$\begin{array}{ll}\underset{\mathbf{x}}{\text{minimize}} & S_\epsilon(\mathbf{x}) = \sum_{i=1}^{m}\sqrt{(\mathbf{C}\mathbf{B}\mathbf{x})_i^2 + \epsilon^2}\\ \text{subject to} & \Phi\mathbf{x} = \mathbf{f},\quad \mathbf{x} \in [0,1]^n.\end{array} \tag{9}$$

Using results from Beck [16], we have the nonasymptotic and asymptotic convergence rates of the IRLS algorithm. Theorem 2.4 predicts that the IRLS sequence starts with a linear rate of convergence when the term $\left(\frac{1}{2}\right)^{\frac{T-1}{2}}(S_\epsilon(\mathbf{x}^{(0)}) - S_\epsilon{}^*)$ is dominating, and then it switches to a sublinear rate of convergence. The asymptotic result Theorem 2.5 removes the dependence on the largest eigenvalue of $\mathbf{B}^{\mathrm{T}}\mathbf{C}^2\mathbf{B}$ and is thus data independent. The formal statements of these theorems depend on a number of parameters, such as the parameter R, from Beck's paper that we do not explicitly define due to its intricacy.

Theorem 2.4 (Theorem 4.2 of [16])**.** *Let* $\{\mathbf{x}^{(l)}\}_{l\geq 0}$ *be the sequence generated by the IRLS method with smoothing parameter* ϵ*. Then for any* $T \geq 2$

$$S_\epsilon(\mathbf{x}^{(T)}) - S_\epsilon{}^* \leq \max\left\{\left(\frac{1}{2}\right)^{\frac{T-1}{2}}(S_\epsilon(\mathbf{x}^{(0)}) - S_\epsilon{}^*), \frac{8\lambda_{\max}(\mathbf{B}^{\mathrm{T}}\mathbf{C}^2\mathbf{B})R^2}{\epsilon(T-1)}\right\}$$

where $S_\epsilon{}^*$ *is the optimal value of (9), and R is the diameter of the level set of (2) (see (3.8) in [16]).*

Theorem 2.5 (Theorem 4.8 of [16])**.** *Let* $\{\mathbf{x}^{(l)}\}_{l\geq 0}$ *be the sequence generated by the IRLS method with smoothing parameter* ϵ*. Then there exists* $K > 0$ *such that for all* $T \geq K + 1$

$$S_\epsilon(\mathbf{x}^{(T)}) - S_\epsilon{}^* \leq \frac{48R^2}{\epsilon(T-K)}$$

Regarding the convergence property of the iterates $\{\mathbf{x}^{(l)}\}_{l\geq 0}$, we have the following theorem.

Theorem 2.6 (Lemma 4.5 of [16])**.** *Let* $\{\mathbf{x}^{(l)}\}_{l\geq 0}$ *be the sequence generated by the IRLS method with smoothing parameter* ϵ*. Then*

(i) *any accumulation point of* $\{\mathbf{x}^{(l)}\}_{l\geq 0}$ *is an optimal solution of (9),*
(ii) *for any* $i \in \{1,\ldots,m\}$*, the sequence* $\{|(\mathbf{C}\mathbf{B}\mathbf{x}^{(l)})_i|\}_{l\geq 0}$ *converges.*

We denote by $\mathcal{N}(s)$ the neighborhood of the source node s, and $\mathcal{N}(t)$ the neighborhood of the sink node t. Then from Theorem 2.6(ii), we can infer that the sequence $\{\mathbf{x}_u^{(l)}\}_{l\geq 0}$ converges for any $u \in \mathcal{N}(s) \cup \mathcal{N}(t)$. This is because $\mathbf{C}$ is diagonal, and $0 = \mathbf{x}_t^{(l)} \leq \mathbf{x}_u^{(l)} \leq \mathbf{x}_s^{(l)} = 1$ according to Proposition 2.2.

2.3. A Cheeger-type inequality

In Section 2.1 we have shown that the IRLS algorithm relates the undirected s–t min-cut problem to solving a sequence of reduced Laplacian systems. Here we prove a Cheeger-type inequality that characterizes the undirected s–t min-cut by the second finite generalized eigenvalue of a particular matrix pencil. The following theorem plays no role in our derivation

and can easily be skipped. However, we proved it while seeking more rigorous stopping criteria for the numerical iterations essential to our procedure. We believe it may play a role in that capacity in the future. We start with some notations and definitions. Let $C = 2\sum_{\{u,v\}\in\mathcal{E}} c(\{u,v\})$ i.e., twice the total edge capacities of $\mathcal{G}$. We define a node-volume function $\mathbf{d}(\cdot)$ on $\mathcal{V}$ to be

$$\mathbf{d}(u) = \begin{cases} C & u = s \text{ or } u = t \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

The volume of a subset $\mathcal{S} \subset \mathcal{V}$ is defined by $\mathrm{vol}(\mathcal{S}) = \sum_{u\in\mathcal{S}} \mathbf{d}(u)$. As usual we define $\phi(\mathcal{S}) = \frac{\mathrm{cut}(\mathcal{S},\bar{\mathcal{S}})}{\min\{\mathrm{vol}(\mathcal{S}),\mathrm{vol}(\bar{\mathcal{S}})\}}$, and the expansion parameter to be $\phi = \min_{\mathcal{S}\subset\mathcal{V}} \phi(\mathcal{S})$. We have $\phi(S) < \infty$ if and only if $(\mathcal{S}, \bar{\mathcal{S}})$ is an s–t cut. And $\phi(S) = \phi$ if and only if $(\mathcal{S}, \bar{\mathcal{S}})$ is an s–t min-cut. Let $\mathbf{D}$ be the $n \times n$ matrix with its main diagonal being $\mathbf{d}$, and let $\mathbf{L}$ be the Laplacian of $\mathcal{G}$. We consider the generalized eigenvalue problem of the matrix pencil ($\mathbf{L}$, $\mathbf{D}$)

$$\mathbf{L}\mathbf{x} = \lambda\mathbf{D}\mathbf{x}. \tag{11}$$

Because $\mathbf{D}$ is of rank 2, the pencil ($\mathbf{L}$, $\mathbf{D}$) has only two finite generalized eigenvalues (see Definition 3.3.1 in [20]). We denote them by $\lambda_1 \le \lambda_2$. We state a Cheeger-type inequality for undirected s–t min-cut as follows.

Theorem 2.7. *Let ϕ be the min-cut and let λ_2 be the non-zero eigenvalue of (11), then $\frac{\phi^2}{2} \le \lambda_2 \le 2\phi$.*

We defer this proof to the Appendix because it is a straightforward generalization of the proof of Theorem 1 in [21]. It also follows from a recent generalized Cheeger inequality by [22].

3. Parallel implementation of the IRLS algorithm

In this section, we describe our parallel implementation of the IRLS algorithm. In addition, we also detail the rounding procedure we use to obtain an approximate s–t min-cut from the voltage vector $\mathbf{x}^{(T)}$ output from the IRLS iterations.

3.1. A parallel iterative solver for the sequence of reduced Laplacian systems

The main focus of the IRLS algorithm is to solve a sequence of reduced Laplacian systems for $l = 1, \ldots, T$. We keep $\widetilde{\mathbf{L}}^{(l)} = \mathbf{Z}^{\mathrm{T}}\mathbf{L}^{(l)}\mathbf{Z}$ and set $\mathbf{b}^{(l)} = -\mathbf{Z}^{\mathrm{T}}\mathbf{L}^{(l)}\mathbf{e}_s$ in the following; note that the system $\widetilde{\mathbf{L}}^{(l)}\mathbf{v} = \mathbf{b}^{(l)}$ is equivalent to (5) as explained in the proof to Proposition 2.2. Because $\widetilde{\mathbf{L}}^{(l)}$ is symmetric positive definite, we can use the preconditioned conjugate gradient (PCG) algorithm to solve it in parallel [23]. For preconditioning, we choose to use the block Jacobi preconditioner because of its high parallelism. Specifically, we extract from $\widetilde{\mathbf{L}}^{(l)}$ a $p \times p$ block diagonal matrix $\widetilde{\mathbf{M}}^{(l)} = \mathrm{diag}(\widetilde{\mathbf{M}}_1^{(l)}, \ldots, \widetilde{\mathbf{M}}_p^{(l)})$, where p is the number of processes. We use $\widetilde{\mathbf{M}}^{(l)}$ to precondition solving the linear system of $\widetilde{\mathbf{L}}^{(l)}$. In each preconditioning step of the PCG iteration, we need to solve a linear system involving $\widetilde{\mathbf{M}}^{(l)}$. Because of the block diagonal structure of $\widetilde{\mathbf{M}}^{(l)}$, the p independent subsystems

$$\widetilde{\mathbf{M}}_j^{(l)}\mathbf{y}_j = \mathbf{r}_j \quad j = 1, \ldots, p \tag{12}$$

can be solved in parallel. We consider two strategies to solve the subsystems (12): (I) LU factorization to exactly solve (12); and (II) incomplete LU factorization with zero fill-in (ILU(0)) to approximately solve (12). Given that the nonzero structure of $\widetilde{\mathbf{L}}^{(l)}$ never changes (it is fully determined by the non-terminal graph $\widetilde{\mathcal{G}}$), the symbolic factorization of LU and ILU(0) needs to be done only once. As we will demonstrate in Section 5, applying (I) or (II) highly depends on the application. Given that we are solving a sequence of related linear systems, where the solution $\mathbf{v}^{(l)}$ is used to define $\widetilde{\mathbf{L}}^{(l+1)}$ and $\mathbf{b}^{(l+1)}$, we adopt a *warm start* heuristic. Specifically, we let $\mathbf{v}^{(l)}$ be the initial guess to the parallel PCG iterations for solving the $(l+1)$th linear system. In contrast, *cold start* involves always using the zero initial guess.

3.2. Parallel graph partitioning

Given a fixed number of processes p, we consider the problem of extracting from $\widetilde{\mathbf{L}}^{(l)}$ an effective and efficient block Jacobi preconditioner $\widetilde{\mathbf{M}}^{(l)}$. For this purpose, we impose the following two objectives:

(i) The Frobenius norm ratio $\|\widetilde{\mathbf{M}}^{(l)}\|_{\mathrm{F}}/\|\widetilde{\mathbf{L}}^{(l)}\|_{\mathrm{F}}$ is as large as possible.
(ii) The sizes and number of nonzeros in $\widetilde{\mathbf{M}}_j^{(l)}$, $j = 1, \ldots, p$ are well balanced.

The above two objectives can be formulated as a k-way graph partitioning problem [24,25]. Let $\widetilde{\mathcal{G}}^{(l)}$ be the reweighted non-terminal graph whose edge capacities are determined by the off-diagonal entries of $\widetilde{\mathbf{L}}^{(l)}$. Ideally we need to apply the k-way graph partitioning to $\widetilde{\mathcal{G}}^{(l)}$ for each $l = 1, \ldots, T$. However, according to (7) we observe that large edge capacities tend to be large after reweighting because of the $\mathbf{C}^2$ in (7). This motivates us to reuse graph partitioning. Specifically, we use the parallel graph partitioning package ParMETIS[1] [24] to partition the non-terminal graph $\widetilde{\mathcal{G}}$ into p components with the

[1] http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview .

objective of minimizing the capacitated edge cut, while balancing the volumes of different components. We then use this graph partitioning result for all the following IRLS iterations. The graph partitioning result implies a reordering of the nodes in $\mathcal{V}\backslash\{s,t\}$ such that nodes belonging to the same component are numbered continuously. Let the permutation matrix of this reordering be $\mathbf{P}$. We then extract $\widetilde{\mathbf{M}}^{(l)}$ as the block diagonal submatrix of the reordered matrix $\mathbf{P}\widetilde{\mathbf{L}}^{(l)}\mathbf{P}^{\mathrm{T}}$.

3.3. Graph distribution and parallel graph reweighting

The input graph $\mathcal{G}$ consists of the non-terminal graph $\widetilde{\mathcal{G}}$ and the terminal edges $\mathcal{E}^{\mathcal{T}}$. Let $\widetilde{\mathbf{L}}$ be the graph Laplacian of $\widetilde{\mathcal{G}}$. For distributing $\widetilde{\mathcal{G}}$ among p processes, we partition the reordered matrix $\mathbf{P}\widetilde{\mathbf{L}}\mathbf{P}^{\mathrm{T}}$ into p block rows and assign each block row to one process. The graph partitioning also induces a partition of the terminal edges as $\mathcal{E}^{\mathcal{T}} = \bigcup_{j=1}^{p} \mathcal{E}_j^{\mathcal{T}}$, and each $\mathcal{E}_j^{\mathcal{T}}$ is assigned to one process. Accordingly we also partition and distribute the permuted vectors $\mathbf{P}\mathbf{x}^{(l)}$ and $\mathbf{P}\mathbf{b}^{(l)}$ among p processes. At the lth IRLS iteration, we need to form the reweighted Laplacian (7) using the voltage vector $\mathbf{P}\mathbf{x}^{(l-1)}$. After the p processes have communicated to acquire their needed components of $\mathbf{P}\mathbf{x}^{(l-1)}$, they can form their local block row of $\mathbf{P}\widetilde{\mathbf{L}}^{(l)}\mathbf{P}^{\mathrm{T}}$ and $\mathbf{P}\mathbf{b}^{(l)}$ in parallel. Thus the cost of one parallel graph reweighting is equivalent to that of one parallel matrix–vector multiplication. Note that the k-way graph partitioning usually results in a small number of edges between different components. This characteristic helps to reduce the process communication cost.

3.4. A two-level rounding procedure

Let $\mathbf{x}^{(T)}$ be the voltage vector output from running the IRLS algorithm for T iterations. We consider the problem of converting $\mathbf{x}^{(T)}$ to a binary cut indicator vector. A standard technique for this purpose is sweep cut [17], which is based on sorting the nodes according to $\mathbf{x}^{(T)}$. Here we propose a heuristic rounding procedure, which we refer to as the *two-level rounding procedure*. Empirically we observe the components of $\mathbf{x}^{(l)}$ tend to converge to {0, 1} as l gets larger. We call this effect the *node voltage polarization*. This observation motivates the idea of coarsening the graph $\mathcal{G}$ using $\mathbf{x}^{(T)}$. Given $\mathbf{x}^{(T)}$ and another two parameters γ_0 and γ_1, we partition $\mathcal{V} = \mathcal{S}_0 \cup \mathcal{S}_1 \cup \mathcal{S}_c$ as follows:

$$u \in \begin{cases} \mathcal{S}_0, & \text{if } \mathbf{x}_u^{(T)} \le \gamma_0 \\ \mathcal{S}_1, & \text{if } \mathbf{x}_u^{(T)} \ge \gamma_1 \\ \mathcal{S}_c, & \text{if } \gamma_0 < \mathbf{x}_u^{(T)} < \gamma_1 \end{cases} \tag{13}$$

Intuitively, we coalesce node u with the sink (source) node if $\mathbf{x}_u^{(T)}$ is small (large) enough, and otherwise we leave it alone. Contracting $\mathcal{S}_0$ ($\mathcal{S}_1$) to a single node t_c (s_c), we define a coarsened graph $\mathcal{G}_c = (\mathcal{V}_c, \mathcal{E}_c)$ with node set $\mathcal{V}_c = \{s_c, t_c\} \cup \mathcal{S}_c$.

The capacitated edge set $\mathcal{E}_c$ of the coarsened graph is defined according to the following cases:

- $\{u, v\} \in \mathcal{E}_c$ with capacity $c(\{i, j\})$ if $u, v \in \mathcal{S}_c$ such that $\{u, v\} \in \mathcal{E}$.
- $\{s_c, u\} \in \mathcal{E}_c$ with capacity $\sum_{v \in \mathcal{S}_1} c(\{u, v\})$.
- $\{t_c, u\} \in \mathcal{E}_c$ with capacity $\sum_{v \in \mathcal{S}_0} c(\{u, v\})$.
- $\{s_c, t_c\} \in \mathcal{E}_c$ with capacity $\sum_{u \in \mathcal{S}_1} \sum_{v \in \mathcal{S}_0} c(\{u, v\})$.

An s–t cut on $\mathcal{G}_c$ induces an s–t cut on $\mathcal{G}$. The idea of the two-level rounding procedure is to solve the undirected s–t min-cut problem on the coarsened graph $\mathcal{G}_c$ using a combinatorial max-flow/min-cut solver, e.g., [7]; and then get the corresponding s–t cut on $\mathcal{G}$. Regarding the quality of such an acquired s–t cut on $\mathcal{G}$, we have the following proposition.

Proposition 3.1. *Let $\mathcal{V} = \mathcal{S}_0 \cup \mathcal{S}_1 \cup \mathcal{S}_c$ be the partition generated by coarsening using $\mathbf{x}^{(T)}$ and parameters γ_0, γ_1. If there is an s–t min-cut on $\mathcal{G}$ such that $\mathcal{S}_1$ is contained in the source side, and $\mathcal{S}_0$ is contained in the sink side, then an s–t min-cut on $\mathcal{G}_c$ induces an s–t min-cut on $\mathcal{G}$.*

Choosing the values of γ_0 and γ_1 is critical to the success of the two-level rounding procedure. On the one hand, we want to set their values conservatively so that not many nodes in $\mathcal{S}_0$ and $\mathcal{S}_1$ are coalesced to the wrong side. On the other hand, we want to set their values aggressively so that the size of $\mathcal{G}_c$ is much smaller than $\mathcal{G}$. We find a good trade-off is achieved by clustering the components of $\mathbf{x}^{(T)}$. Let the two centers returned from K-means on $\mathbf{x}^{(T)}$ (with initial guesses 0.1 and 0.9) be c_0 and c_1, then we let $\gamma_0 = c_0 + 0.05$ and $\gamma_1 = c_1 - 0.05$.

We summarize the algorithm PIRMCut in Algorithm 1. Note the main sequential part of PIRMCut is the rounding procedure (either sweep cut or our two-level rounding).

4. Related work

Approaches to s–t min-cut based on ℓ_1-minimization are common. Bhusnurmath and Taylor [26], for example, transform the ℓ_1-minimization problem to its linear programming (LP) formulation, and then solve the LP using an interior point method with logarithmic barrier functions. Similar to our IRLS approach, their interior point method also leads to solving a sequence of Laplacian systems. In that vein, [4] develops the then theoretically fastest approximation algorithms for s–t

Algorithm 1 PIRMCut $(\mathcal{G}, s, t, p, T)$.

Require: A capacitated and undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the source node s, the sink node t, the number of processes p, and the number of IRLS iterations T.

Ensure: An approximate s-t min-cut on $\mathcal{G}$.

1: Let $\widetilde{\mathcal{G}}$ be the non-terminal graph and $\mathcal{E}^{\mathcal{T}}$ be the terminal edges. Use ParMETIS to partition $\widetilde{\mathcal{G}}$ into p components. Reorder and distribute $\widetilde{\mathcal{G}}$ and $\mathcal{E}^{\mathcal{T}}$ to p processes accordingly.
2: Initialize $\mathbf{x}^{(0)}$ to be the solution to (5) with $\mathbf{W}^{(0)} = \mathbf{C}$ using the parallel PCG solver.
3: **for** $l = 1, \ldots, T$ **do**
4: Execute the parallel IRLS algorithm by alternating between parallel graph reweighting and solving the reduced Laplacian system $\widetilde{\mathbf{L}}^{(l)}\mathbf{v} = \mathbf{b}^{(l)}$ using the parallel PCG solver.
5: **end for**
6: Gather the voltage vector $\mathbf{x}^{(T)}$ to the root process.
7: The root process applies a rounding procedure on $\mathbf{x}^{(T)}$ to get an approximate s-t min-cut on $\mathcal{G}$.

max-flow/min-cut problems in undirected graphs. Their algorithm involved a sequence of electrical flow problems defined via a multiplicative weight update. We are still trying to formalize the relation between our IRLS update and the multiplicative weight update used in [4]. More recent progress of employing electrical flows to approximately solve the undirected s–t max-flow/min-cut problems to achieve better complexity bounds include [5,6]. In contrast, the main purpose of this paper is to implement a parallel s–t min-cut solver using this Laplacian paradigm, and evaluate its performance on parallel computing platforms. Beyond the electrical flow paradigm, which always satisfies the flow conservation constraint but relaxes the capacity constraint, the fastest combinatorial algorithms, like push-relabel and its variants [8,27] and pseudoflow [10], relaxes the flow conservation constraint but satisfying the capacity constraint. However, parallelizing these methods is difficult.

Recent advances on nearly linear time solver for Laplacian and symmetric diagonally dominant linear systems are [28–30]. We do not incorporate the current available implementations of [29] and [30] into PIRMCut because they are sequential. Instead we choose the block Jacobi preconditioned PCG because of its high parallelism. We notice an idea similar to our two-level rounding procedure that is called *flow-based rounding* has been proposed in [31]. This idea was later rigorously formalized as the FlowImprove algorithm [1].

5. Experimental results

In this section, we describe experiments with PIRMCut on several real world graphs as well as synthetic graphs. These experiments serve to empirically demonstrate the properties of PIRMCut, and also evaluate its performance on different undirected s–t min-cut problems. Given the focus of this paper, we only consider undirected s–t min-cut problems with floating-point edge capacities in the following experiments. In fact most interesting applications of undirected s–t min-cut produce floating-point valued problems, e.g., FlowImprove [1], image segmentation [2], MRI analysis [3], and energy minimization in MRF [32]. Thus, we do not include those state-of-the-art serial max-flow/min-cut solvers for integer valued problems in the following experiments, like `hipr-v3.4`[2] [8], `ibfs`[3] [9], and `pseudo-max-3.23`[4] [10]. For the parallel max-flow/min-cut solvers that we know to be available online, we checked that either they did not work for floating-point valued instances like `d_maxflow_ar-1.1`[5] [33]; or they only work for graphs with repeated regular structures like `regionpushrelabel-v1.08`[6] [34] or graphs with nodes' coordinates information like `mpimaxflow`[7] [35]. To the best of our knowledge, besides our PIRMCut, the Boykov–Kolmogorov solver[8] [7] is the only max-flow/min-cut solver that can handle both floating-point edge capacities and undirected graphs with arbitrary structures. We refer to it as the B–K Solver in the following and use it as a baseline solver with which to compare our PIRMCut solver.

5.1. Data sets

The graphs we use to evaluate PIRMCut are shown in Table 1. The road networks are from the University of Florida Sparse Matrix collection [36]. From these road networks, we generate their undirected s–t min-cut instances using the FlowImprove algorithm [1] starting from a geometric bisection. The N-D grid graphs are the segmentation problem instances from the University of Western Ontario max-flow datasets[9] [2]. The ALE graphs [37] are downloaded from

[2] http://www.igsystems.com/hipr/download.html .
[3] http://www.cs.tau.ac.il/~sagihed/ibfs/ .
[4] http://riot.ieor.berkeley.edu/Applications/Pseudoflow .
[5] http://cmp.felk.cvut.cz/~shekhovt/d_maxflow/ .
[6] http://vision.csd.uwo.ca/code/ .
[7] http://www.maths.lth.se/matematiklth/personal/petter/cppmaxflow.php .
[8] http://pub.ist.ac.at/~vnk/software.html .
[9] http://vision.csd.uwo.ca/maxflow-data .

Table 1
All graphs are undirected and we report the size of the non-terminal graph.

Graph	$\lvert\widetilde{\mathcal{V}}\rvert$	$\lvert\widetilde{\mathcal{E}}\rvert$	Kind
usroads-48	126,146	161,950	
asia_osm	11,950,757	12,711,603	Road networks
euro_osm	50,912,018	54,054,660	
adhead.n26c100	12,582,912	163,577,856	
bone.n26c100	7,798,784	101,384,192	N-D grid graphs
liver.n26c100	4,161,600	54,100,800	
graph_2007_000123_5	191,438	3,109,801	
graph_2007_000123_12	191,582	3,498,515	Automatic Labeling Environment (ALE)
graph_2007_000123_33	190,483	3,096,823	
wash-rlg-long.n2048.1163	131,074	393,024	
wash-rlg-long.n4096.1163	262,146	786,240	Wash-Long
wash-rlg-long.n8192.1163	524,290	1,572,672	
wash-rlg-wide.n2048.1163	131,074	387,072	
wash-rlg-wide.n4096.1163	262,146	774,144	Wash-Wide
wash-rlg-wide.n8192.1163	524,290	1,548,288	

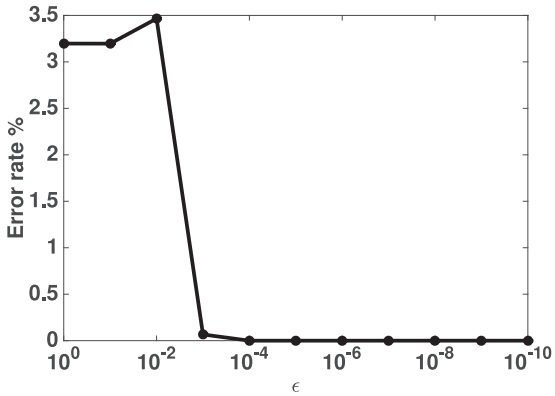

Fig. 1. The error rates of the solutions produced by IRLS with sweep cut, for a sequence of ϵ values.

http://ttic.uchicago.edu/~dbatra/research/mfcomp/#data. The Wash-Long and Wash-Wide graphs are synthetic graphs in the DIMACS families [38] and they are available on http://www.cs.tau.ac.il/~sagihed/ibfs/benchmark.html. For those integer valued s–t min-cut instances, we convert the edge capacities to be floating-point valued by adding uniform random numbers in [0, 1]. All directed graphs are converted to undirected graphs.

5.2. The effect of ϵ

The smoothed objective function $S_\epsilon(\mathbf{x})$ in (9) is continuous in ϵ. Thus, when $\epsilon \to 0$, in theory an optimal solution to (9) would converge to an optimal solution to the ℓ_1-minimization problem (1). Experimentally we demonstrate the effect of the smoothing parameter ϵ on the solution produced by the IRLS algorithm using the graph of usroads-48. We choose the sequence of ϵ values to be $1, 10^{-1}, \ldots, 10^{-10}$. For each value of ϵ, we run the IRLS algorithm for $T = 500$ iterations on a single MPI process (note that the reduced Laplacian systems are all solved directly in this case). Then we apply sweep cut as the rounding procedure to generate the integral solution, which is denoted as $\mathbf{x}^\epsilon$. We denote the solution output by the B–K Solver on the same graph as $\mathbf{x}^*$, which is an integral optimal solution to (1). As a measure of the difference between $\mathbf{x}^\epsilon$ and $\mathbf{x}^*$, we define the following error rate of partitioning the nodes:

$$\text{Error rate} = \frac{|\{u : \mathbf{x}^\epsilon_u \neq \mathbf{x}^*_u\}|}{n}$$

In Fig. 1, we plot the error rates (in percentage) for the sequence of ϵ values. We can see that when $\epsilon = 10^{-3}$, $\mathbf{x}^\epsilon$ is already very close to $\mathbf{x}^*$ (with an error rate of 0.07%). And from $\epsilon = 10^{-4}$ on, IRLS with sweep cut always produces the optimal min-cut solution. We also note that the error rate increases when ϵ decreases from 10^{-1} to 10^{-2}, although we have checked that the corresponding cut value does decreases.

5.3. The effect of warm starts

On the graph of usroads-48, we demonstrate the benefit of the warm start heuristic described in Section 3.1. We set the smoothing parameter $\epsilon = 10^{-6}$ and run the IRLS algorithm for 50 iterations on 4 MPI processes. For each IRLS iterate,

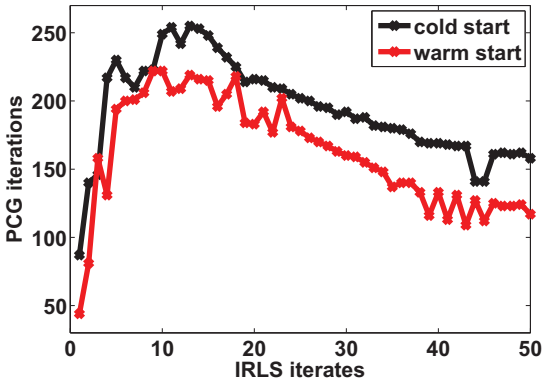

Fig. 2. The number of PCG iterations to reach a residual of 10^{-3} is reduced by roughly 20% by using warm starts for this sequence of Laplacian systems.

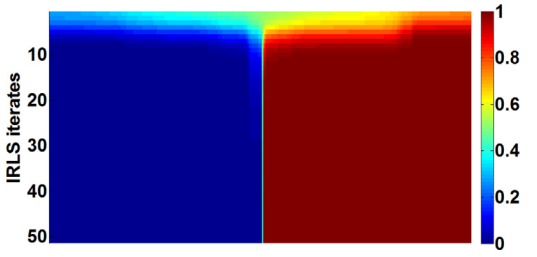

Fig. 3. Node voltage polarization. Each row is a sorted voltage vector $\mathbf{x}^{(l)}$.

we set the maximum number of PCG iterations to be 300, and the stopping criterion in relative residual to be 10^{-3}. In Fig. 2 we plot the number of PCG iterations of using warm starts and cold starts. It is apparent that for most IRLS iterates, warm starts help to reduce the number of needed PCG iterations significantly, especially for later IRLS iterates. Another interesting phenomenon we observe from Fig. 2 is that the difficulty of solving the reduced Laplacian system increases dramatically during the first several IRLS iterates, and then decreases later on. A possible explanation is that the IRLS algorithm makes faster progress during the early iterates.

5.4. Effect of node voltage polarization

Continuing with `usroads-48` as an example, we demonstrate node voltage polarization, which motivates the idea of two-level rounding procedure (see Section 3.4). We plot a heatmap of $\mathbf{x}^{(l)}$ (after sorting its components) for $l = 0, \ldots, 50$ in Fig. 3. It is apparent that the polarization gets emphasized as l becomes larger. We also see from Fig. 3 that the values of $\mathbf{x}^{(l)}$ change quickly for $l \leq 10$, which reinforces the observation of the IRLS algorithm making faster progress early.

5.5. Performance evaluation of PIRMCut

Our parallel implementation of PIRMCut is written in C++, and it is purely based on MPI, no multi-threading is exploited. In particular, the parallel PCG solver with block Jacobi preconditioner is implemented using the PETSc[10] package [39]. For implementing the two-level rounding procedure in PIRMCut we use the B–K Solver. All the results reported in the following are generated using: smoothing parameter $\epsilon = 10^{-6}$, number of IRLS iterations $T = 50$, warm start heuristic, and 50 PCG iterations at maximum. For solving the subsystems (12) we use exact LU factorization by UMFPACK [40] with the only exception that on the N-D grid graphs in Table 1 we use ILU(0) in PETSc [39]. We evaluate the empirical performance of PIRMCut on all the graphs in Table 1 except `usroads-48`.

We use the output of the B–K Solver to evaluate the quality of the approximate s–t min-cut achieved by PIRMCut. Specifically, we denote by μ^* the s–t min-cut value output from B–K Solver, and μ the s–t min-cut value output from PIRMCut. Then we compute the relative approximation ratio

$$\delta = |\mu - \mu^*|/\mu^* \tag{14}$$

According to Algorithm 1 of PIRMCut, the IRLS algorithm is followed by a rounding procedure that is serial in nature. Thus, for studying parallel scalability, we focus on the IRLS iterations. We conduct experiments on both a distributed

[10] http://www.mcs.anl.gov/petsc/ .

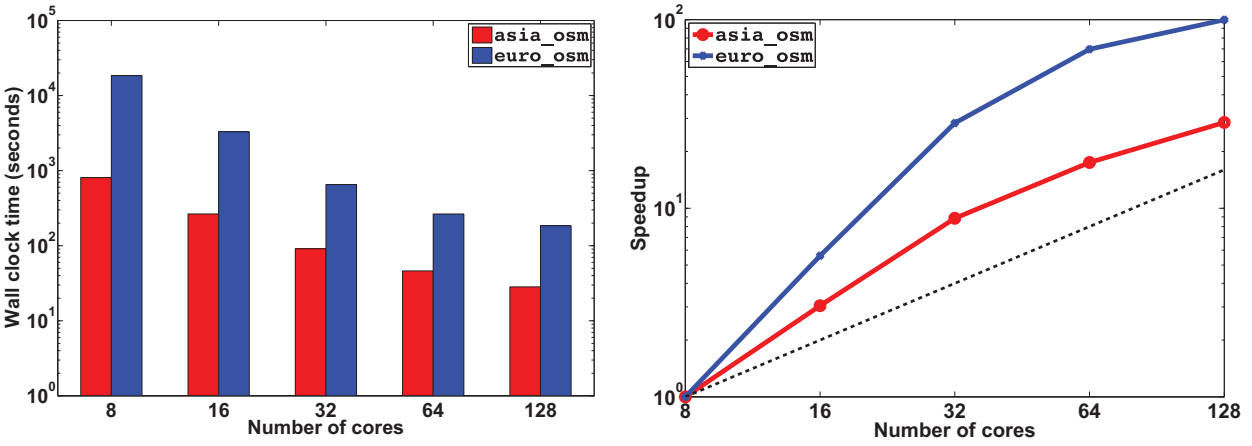

Fig. 4. Parallel scalability of the IRLS iterations of PIRMCut on road networks on the distributed memory platform.

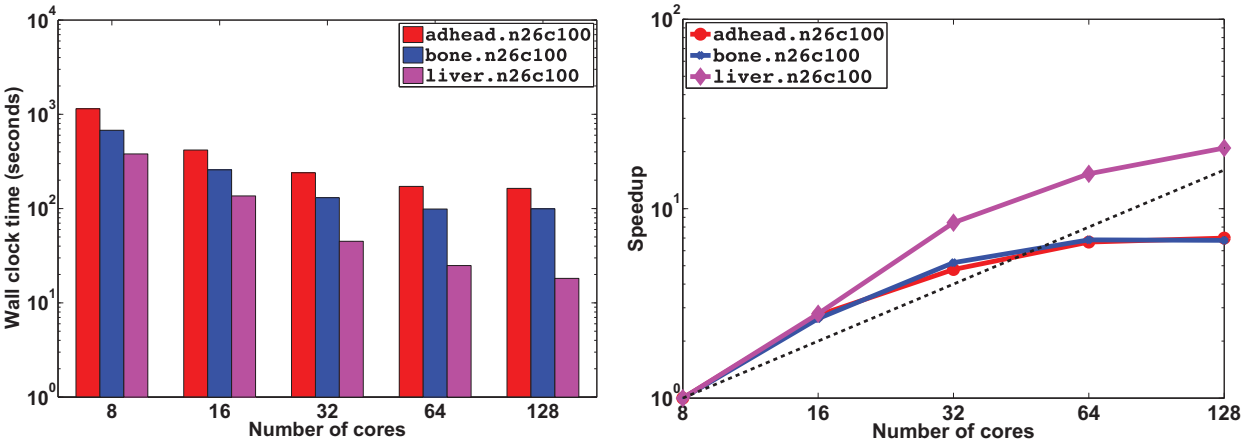

Fig. 5. Parallel scalability of the IRLS iterations of PIRMCut on N-D grid graphs on the distributed memory platform.

memory platform and a shared memory platform, with the results reported in Sections 5.5.1 and 5.5.2 respectively. Our purpose is not to compare the pros and cons of different architectures for PIRMCut, but to get a complete picture of PIRMCut empirical performance. It is known that when the number of MPI processes p increases, the quality of the iterates $\mathbf{x}^{(l)}$, $l = 1, \ldots, T$, generated by the IRLS iterations could decrease because the block Jacobi preconditioner becomes weaker with an increasing p. However, we find that the quality of the approximate solution produced by the entire PIRMCut pipeline is quite insensitive to the number of MPI processes. In fact on both the distributed memory and the shared memory platforms, for each of the graphs reported in Sections 5.5.1 and 5.5.2, the relative approximation ratio δ achieved by PIRMCut using the two-level rounding procedure under different number of MPI processes are of almost the same order of magnitude. It is possible that the adoption of warm start diminishes the effect of the block Jacobi preconditioner becoming weaker when p increases. Another possibility is that the two-level rounding procedure is not very sensitive to the accuracy of $\mathbf{x}^{(T)}$. In essence, $\mathbf{x}^{(T)}$ just need to be enough accurate for $\mathcal{S}_0$ and $\mathcal{S}_1$ in (13) to be identified.

5.5.1. Scalability results on a distributed memory platform

We first study the parallel scalability of the IRLS iterations on a distributed memory machine consists of 8 nodes with each node having 24 cores (Intel Xeon E5-4617 processor) and 64 GB memory. When executing PIRMCut, we utilize all the nodes with each node running the same number of MPI processes. The timing and speedup results on different kinds of graphs are shown in Figs. 4–8.

In Fig. 4, we see on the road networks the speedup is superlinear until $p = 128$. This happens because, as p gets large, the total work of applying the block Jacobi preconditioner decreases dramatically. In Fig. 5, on two of three N-D grid graphs the linear speedup stops after $p = 64$. One reason is that the N-D grid graphs are denser than the road networks (see Table 1), which would incur high communication cost when p is large. Another reason is that the use of ILU(0) on N-D grid graphs makes the work reduction not that critical after p is large enough. In contrast to the large road networks and N-D grid graphs, on the ALE, Wash-Long and Wash-Wide graphs, the speedups stagnate or even decrease immediately after $p = 16$

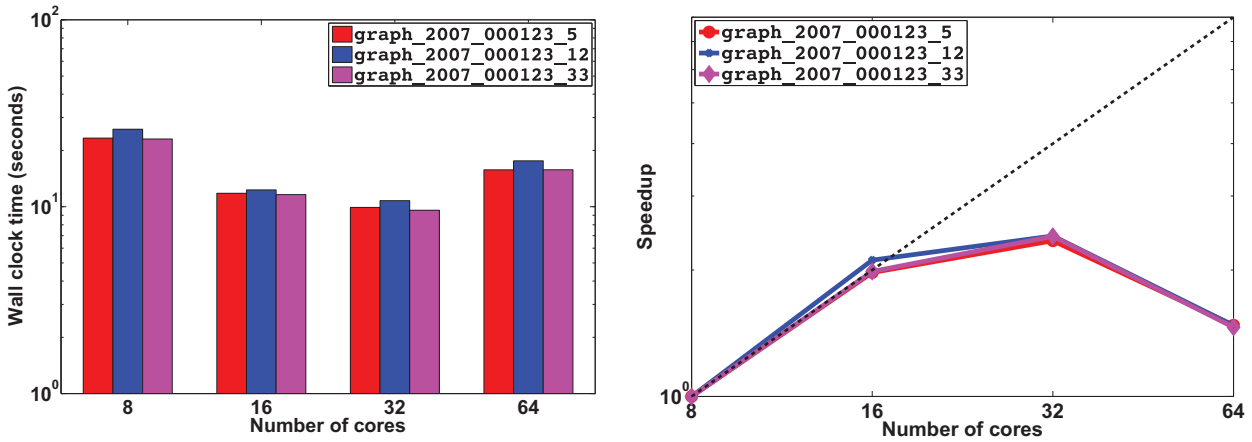

Fig. 6. Parallel scalability of the IRLS iterations of PIRMCut on ALE graphs on the distributed memory platform.

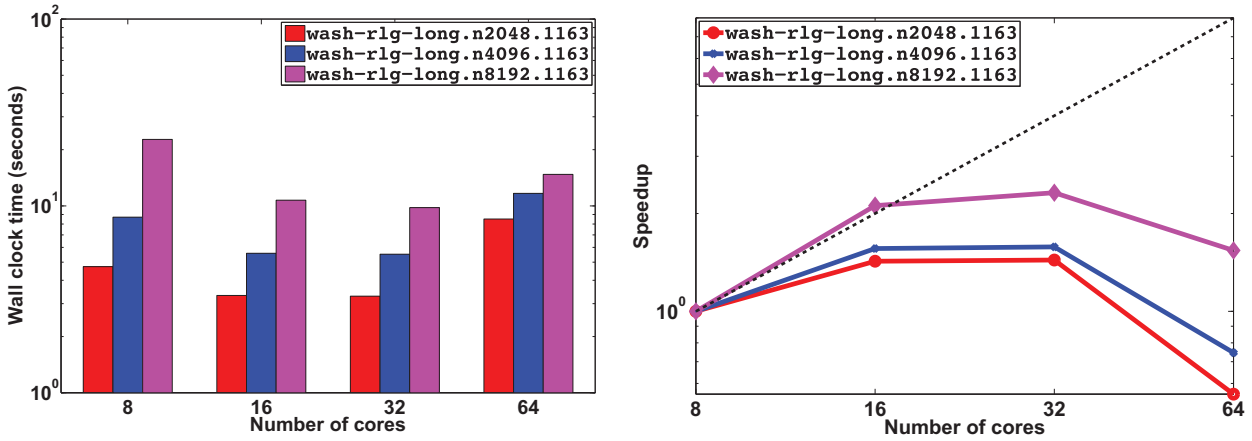

Fig. 7. Parallel scalability of the IRLS iterations of PIRMCut on Wash-Long graphs on the distributed memory platform.

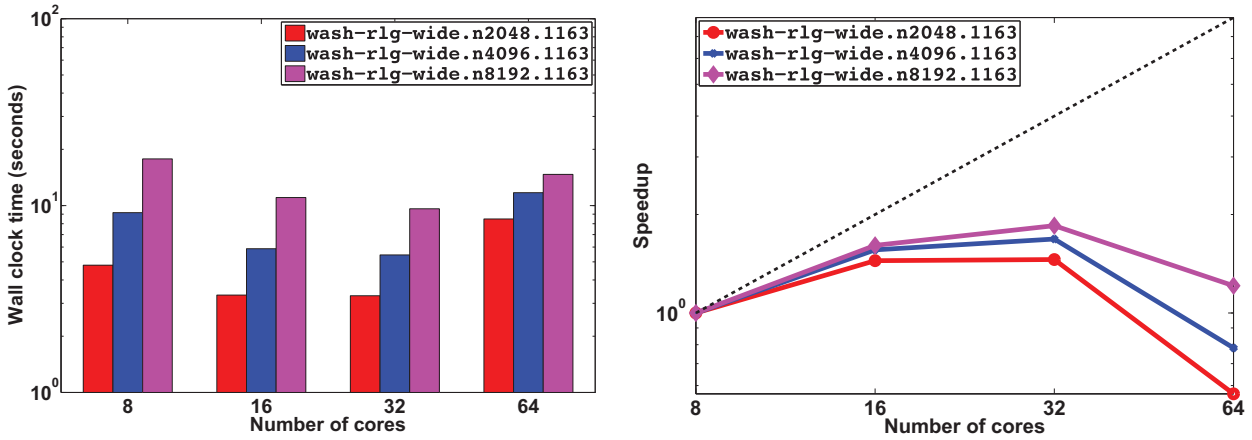

Fig. 8. Parallel scalability of the IRLS iterations of PIRMCut on Wash-Wide graphs on the distributed memory platform.

as shown in Figs. 6–8. This is understandable given the much smaller sizes of these graphs (see Table 1) and relatively small amount of work per process. Thus, over-partitioning them leads to excessively high communication cost that quickly prohibits speedup.

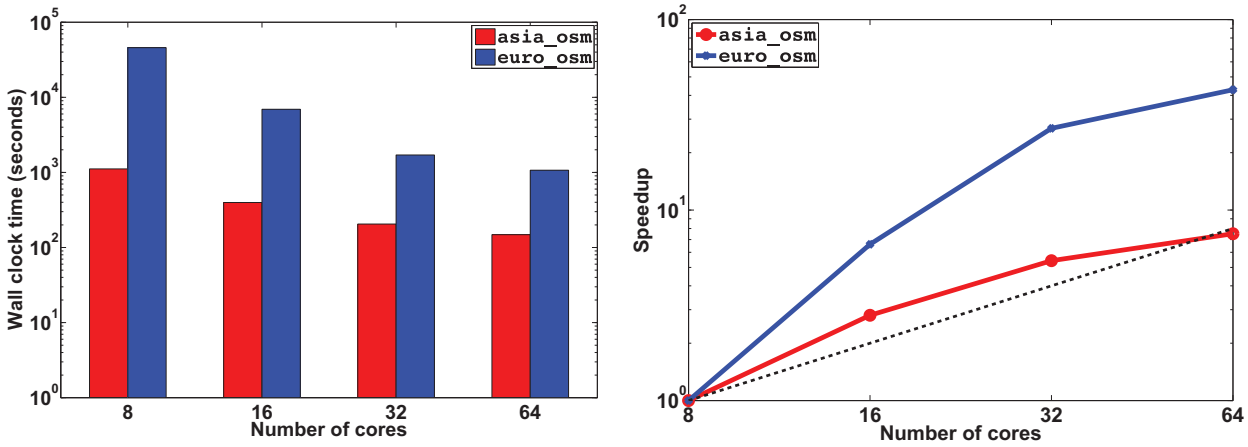

Fig. 9. Parallel scalability of the IRLS iterations of PIRMCut on road networks on the shared memory platform.

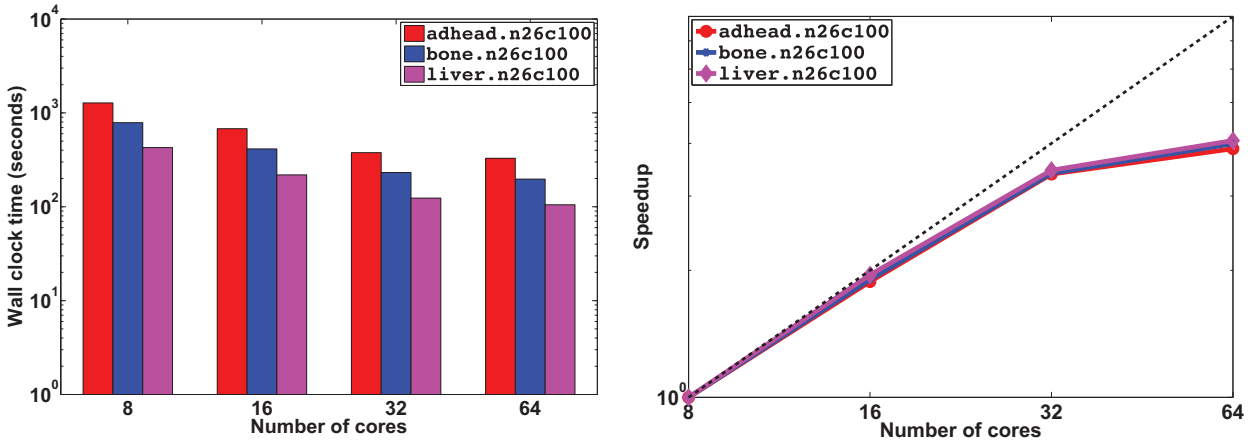

Fig. 10. Parallel scalability of the IRLS iterations of PIRMCut on N-D grid graphs on the shared memory platform.

5.5.2. Scalability results on a shared memory platform

We then study the parallel scalability of the IRLS iterations on a shared memory machine having 80 cores (Intel Xeon E7-8870 processor) and 128GB memory. The timing and speedup results on different kinds of graphs are shown in Figs. 9–13.

In Fig. 9, we see on the road networks the speedup is superlinear until $p = 64$. This again indicates that on the road networks the computation cost from applying the block Jacobi preconditioner is pretty dominating the communication cost. In Fig. 10, the speedup on the N-D grid graphs is almost linear until $p = 32$. One possible reason for the loss of linear speedup when doubling $p = 32$ to $p = 64$ is that at most 40 cores are directly connected through the Intel QPI links on our shared memory machine. Similar to the cases on the distributed memory machine, the speedups on the ALE, Wash-Long and Wash-Wide graphs stagnate after $p = 16$ as shown in Figs. 11–13. However, the slowdown of speedups on these graphs is less severe on the shared memory machine.

5.5.3. Comparison with the B–K Solver

We compare the performance of PIRMCut with that of B–K Solver. The B–K Solver is a serial code, and we run it on one core on both the distributed and shared memory machines.

In Table 2, we report the total time taken by PIRMCut (T_{PC}), the total time taken by B–K Solver ($T_{\text{B-K}}$), the speed improvement of PIRMCut over B–K Solver ($T_{\text{B-K}}/T_{\text{PC}}$), and the relative approximation ratio of PIRMCut from using the two-level rounding procedure (δ). For PIRMCut, its total time in T_{PC} contains the time of the two-level rounding procedure except on the graph `wash-rlg-wide.n8192.1163`. It is because on `wash-rlg-wide.n8192.1163`, the two-level rounding procedure did not terminate due to the non-termination of the B–K Solver on the coarsened graph within our time limit of 4 h of wall clock time.

From the column of $T_{\text{B-K}}/T_{\text{PC}}$ in Table 2, we see that PIRMCut achieves significant speed improvement over B–K Solver on most of the graphs. Specifically, on `graph_2007_000123_5` PIRMCut is 200 times faster than the B–K Solver on

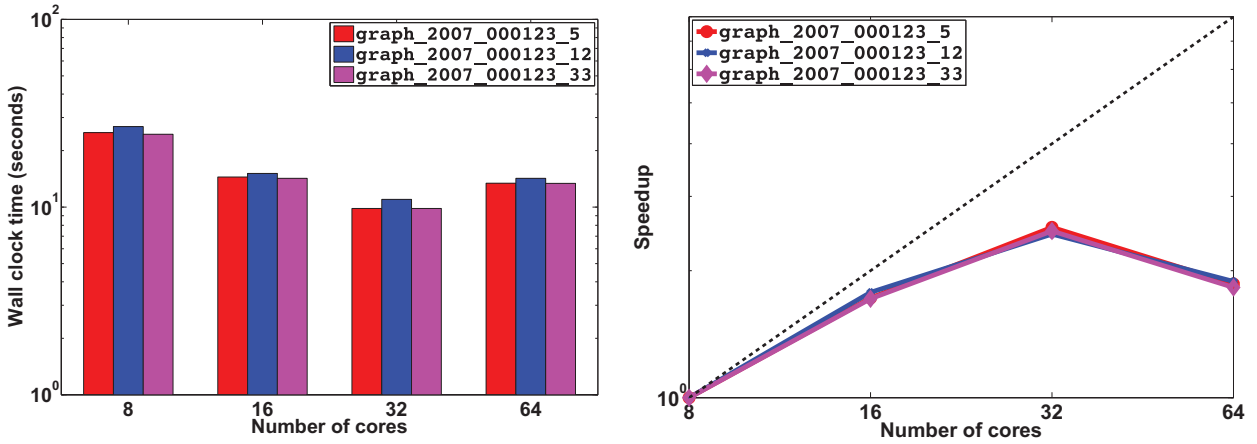

Fig. 11. Parallel scalability of the IRLS iterations of PIRMCut on ALE graphs on the shared memory platform.

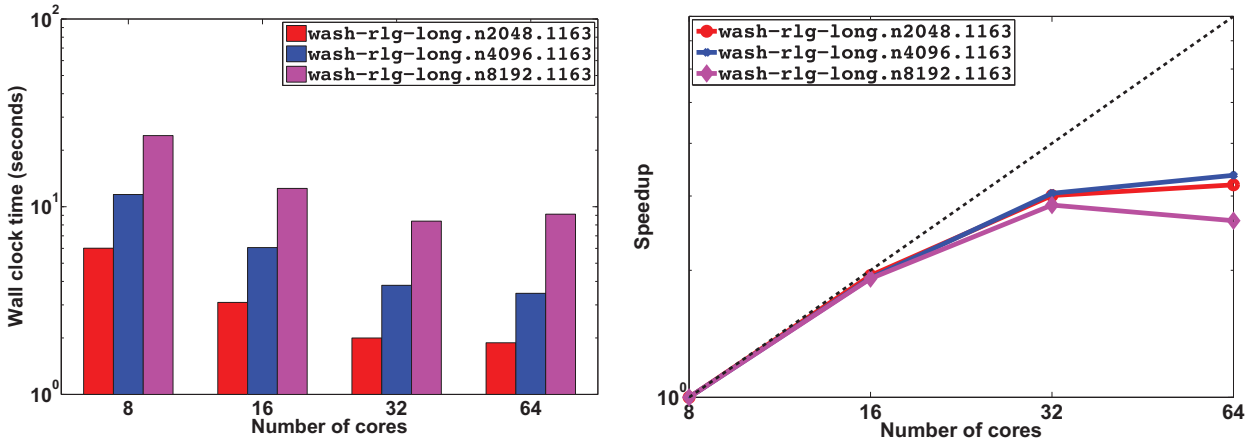

Fig. 12. Parallel scalability of the IRLS iterations of PIRMCut on Wash-Long graphs on the shared memory platform.

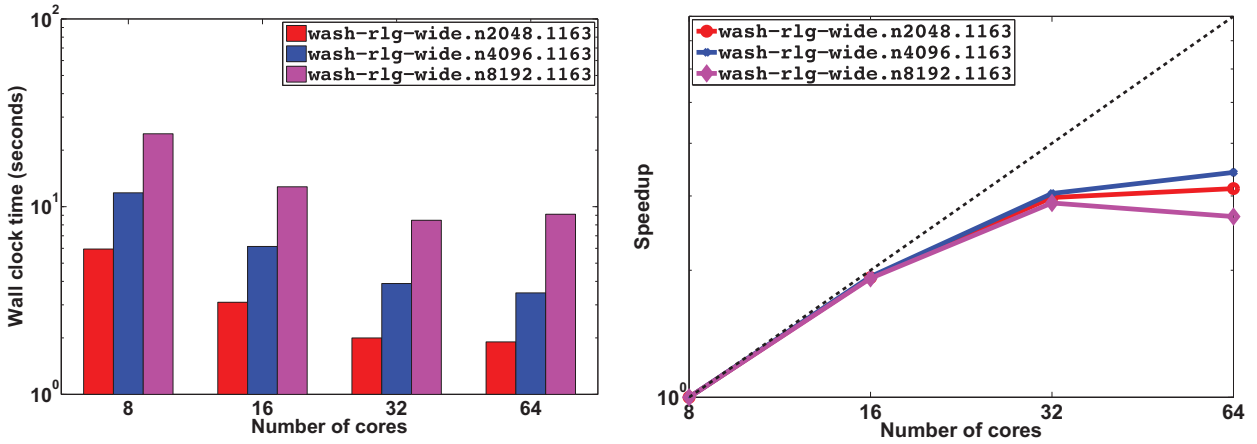

Fig. 13. Parallel scalability of the IRLS iterations of PIRMCut on Wash-Wide graphs on the shared memory platform.

Table 2
Performance comparison between PIRMCut and B–K Solver. All the times are in seconds (rounded to integers). For PIRMCut, we report the best total time and the number of cores p used.

Graph	Platform	p	T_{PC}	$T_{B\text{-}K}$	$T_{B\text{-}K}/T_{PC}$	δ
asia_osm	Distributed memory	128	49	1465	29.8	3.4×10^{-5}
	Shared memory	64	176	2158	12.3	3.0×10^{-5}
euro_osm	Distributed memory	128	301	3102	10.3	6.8×10^{-5}
	Shared memory	64	1248	4715	3.8	7.4×10^{-5}
adhead.n26c100	Distributed memory	128	167	555	3.3	7.0×10^{-14}
	Shared memory	64	332	668	2	7.7×10^{-14}
bone.n26c100	Distributed memory	64	101	215	2.1	2.7×10^{-14}
	Shared memory	64	200	248	1.2	2.7×10^{-14}
liver.n26c100	Distributed memory	128	88	294	3.3	9.0×10^{-4}
	Shared memory	64	197	325	1.6	8.8×10^{-4}
graph_2007_000123_5	Distributed memory	32	22	4404	200	4.6×10^{-3}
	Shared memory	32	28	2175	77.7	4.5×10^{-3}
graph_2007_000123_12	Distributed memory	32	73	202	2.8	2.3×10^{-3}
	Shared memory	16	74	197	2.7	1.8×10^{-3}
graph_2007_000123_33	Distributed memory	32	16	68	4.3	2.0×10^{-16}
	Shared memory	32	20	83	4.2	2.0×10^{-16}
wash-rlg-long.n2048.1163	Distributed memory	32	3	50	16.7	7.0×10^{-2}
	Shared memory	64	2	84	42	7.0×10^{-2}
wash-rlg-long.n4096.1163	Distributed memory	32	6	199	33.2	7.0×10^{-2}
	Shared memory	64	4	340	85	7.0×10^{-2}
wash-rlg-long.n8192.1163	Distributed memory	32	11	764	69.5	7.0×10^{-2}
	Shared memory	32	9	1303	145	7.0×10^{-2}
wash-rlg-wide.n2048.1163	Distributed memory	32	29	457	15.8	1.4×10^{-2}
	Shared memory	64	32	571	17.8	1.4×10^{-2}
wash-rlg-wide.n4096.1163	Distributed memory	64	255	–	–	–
	Shared memory	64	334	–	–	–
wash-rlg-wide.n8192.1163	Distributed memory	32	10	–	–	–
	Shared memory	32	9	–	–	–

the distributed memory machine. It's interesting to observe that although the three graphs graph_2007_000123_5, graph_2007_000123_12, and graph_2007_000123_33 are from the same sequence of related s–t min-cut problems, the times of the B–K Solver on them are dramatically different and unpredictable. In contrast, the times of PIRMCut on these related problems are much more stable and predictable. On the two road networks, PIRMCut also achieves desirable speed improvement over the B–K Solver. Notice that on the shared memory machine, if we extrapolate according to the parallel speedup plot in Fig. 9, the speed improvement of PIRMCut over the B–K Solver could be further continued if we have more cores to utilize. For the DIMACS synthetic graphs Wash-Long and Wash-Wide, although they are of moderate sizes, their floating point valued instances turn out to be very hard for the B–K Solver. It even did not terminate on wash-rlg-wide.n4096.1163 and wash-rlg-wide.n8192.1163 after 4 hours on both the distributed and shared memory machines. In comparison, PIRMCut could solve the Wash-Long and Wash-Wide problems very efficiently. The only family of graphs on which we do not see a significant speed improvement using PIRMCut are the N-D grid graphs. This is understood because the design and implementation of the B–K Solver is highly optimized for vision applications [7]. According to Table 2, PIRMCut enables us to gain significant speed improvement by finding δ-accurate solutions. Moreover, for some graphs PIRMCut could be the only feasible choice given the unpredictable and probably very high execution times of B–K Solver.

For comparing the effectiveness of sweep cut and the two-level rounding procedure, we report their respective relative approximation ratios on different graphs in Table 3. It is shown that for each graph the sweep cut already produces a reasonable approximate solution. In addition, on both the road networks and the N-D grid graphs, the two-level rounding procedure consistently produces much better approximate solutions.

6. Discussion

The PIRMCut algorithm is a step towards our goal of a scalable, parallel s–t min-cut solver for problems with hundreds of millions or billions of edges of which the capacities are floating-point valued. It has much to offer, and demonstrates good scalability and speed improvement over an expertly crafted serial solver.

We have not yet discussed how to take advantage from solving a sequence of related s–t min-cut problems because we have not yet completed our investigation into that setting. That said, we have designed our framework such that solving a sequence of problems is reasonably straightforward. Note that all of the set up overhead of PIRMCut can be amortized when solving a sequence of problems if the graph structure and edge capacities do not change dramatically. The larger issue with a sequence of problems, however, is that our solver only produces δ-accurate solutions. For the applications where a sequence is desired, we need to revisit the underlying methods and see if they can adapt to this type of approximate solutions. This

Table 3
Solution quality of sweep cut and the two-level rounding procedure. This table is based on the result on the distributed memory machine. The result on the shared memory machine is similar.

Graph	Sweep cut	Two-level
asia_osm	2.1×10^{-3}	$\mathbf{3.4 \times 10^{-5}}$
euro_osm	3.2×10^{-2}	$\mathbf{6.8 \times 10^{-5}}$
adhead.n26c100	1.0×10^{-4}	$\mathbf{7.0 \times 10^{-14}}$
bone.n26c100	9.7×10^{-4}	$\mathbf{2.4 \times 10^{-14}}$
liver.n26c100	4.9×10^{-2}	$\mathbf{8.8 \times 10^{-4}}$
graph_2007_000123_5	2.1×10^{-2}	$\mathbf{4.6 \times 10^{-3}}$
graph_2007_000123_12	2.4×10^{-4}	2.3×10^{-3}
graph_2007_000123_33	2.8×10^{-11}	$\mathbf{5.2 \times 10^{-13}}$
wash-rlg-long.n2048.1163	5.4×10^{-2}	7.0×10^{-2}
wash-rlg-long.n4096.1163	5.4×10^{-2}	7.0×10^{-2}
wash-rlg-long.n8192.1163	5.4×10^{-2}	7.0×10^{-2}
wash-rlg-wide.n2048.1163	4.4×10^{-3}	1.4×10^{-2}

setting also offers an opportunity to study the accuracy required for solving a sequence of problems. It is possible that by solving each individual problem to a lower accuracy, it may generate a sequence with a superior final objective value for application purposes. Alternatively, the method may be able to solve the cut problems to a fairly coarse accuracy much more rapidly than existing exact solvers, and these approximations may suffice for many applications. Thus, the goal of our future investigation on a sequence of problems will attempt to solve the sequence faster and better through carefully engineered approximations.

There are also a few further investigations we plan to conduct regarding our framework. First, currently we used a pure MPI based parallel PCG solver for the diagonally dominant Laplacians. Recent work on the nearly linear time solvers for such systems shows the potential for fast runtime, but the parallelization potential is not yet evident in the literature. We wish to explore the possibility of parallelizing these nearly linear time solvers as well as incorporating them into our framework. It is also worthy to pursue a multithreaded implementation of PIRMCut. Given the recent advances in systems and architectures targeted at irregular applications [12–14], a multithreaded implementation of PIRMCut could obtain superior speedups on these platforms.

Second, we wish to make our two-level rounding procedure more rigorous. As we have mentioned, this method is similar to Lang's flow-based rounding for the spectral method [31] that was later formalized in [1]. We hypothesize that a similar formalization will hold here based on which we will be able to create a more principled approach. Furthermore, we suspect that a multi-level extension of this will also be necessary as we continue to work on even larger problems.

Finally, we plan to continue working on improving the analysis of our PIRMCut algorithm in an attempt to formally bound its runtime. This will involve designing more principled stopping criteria based on tighter diagnostics, e.g., from applying our Cheeger-type inequality.

Acknowledgments

This work was partially supported by NSF CAREER award CCF-1149756 and ARO Award 1111691-CNS.

Appendix

We first prove the following characterization of λ_2.

Proposition A.1. *λ_2 is the optimal value of the constrained WLS problem*

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2C}\mathbf{x}^T\mathbf{L}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{x}_s = 1, \quad \mathbf{x}_t = -1 \end{aligned} \tag{A.1}$$

Proof. By inspection, $\lambda_1 = 0$ with $\mathbf{x} = \mathbf{e}$. Thus, λ_2 can be represented by

$$\lambda_2 = \min_{\mathbf{e}^T\mathbf{D}\mathbf{x}=0} \frac{\mathbf{x}^T\mathbf{L}\mathbf{x}}{\mathbf{x}^T\mathbf{D}\mathbf{x}} \tag{A.2}$$

The constraint $\mathbf{e}^T\mathbf{D}\mathbf{x} = 0$ is equivalent to $\mathbf{x}_s + \mathbf{x}_t = 0$. Because the representation (A.2) is invariant to scaling of $\mathbf{x}$, we can constrain $\mathbf{x}_s = 1$ and $\mathbf{x}_t = -1$, which results in $\mathbf{x}^T\mathbf{D}\mathbf{x} = 2C$. □

Note that (A.1) has a form similar to (5), where the main difference is that we use $\{1, -1\}$ to encode the boundary condition. We now prove the Cheeger-type inequality.

Theorem A.2. $\frac{\phi^2}{2} \le \lambda_2 \le 2\phi$

Proof. We first prove the direction of $\lambda_2 \le 2\phi$. Let $(\mathcal{S}, \bar{\mathcal{S}})$ be an s–t min-cut such that $s \in \mathcal{S}$ and $t \in \bar{\mathcal{S}}$. Then $\phi(\mathcal{S}) = \phi$. We define the $\{1, -1\}$ encoded cut indicator vector

$$\mathbf{x}_u = \begin{cases} 1 & \text{if } u \in \mathcal{S} \\ -1 & \text{if } u \in \bar{\mathcal{S}} \end{cases}$$

This vector satisfies $\mathbf{e}^{\mathrm{T}}\mathbf{D}\mathbf{x} = 0$. Then by representation (A.2)

$$\begin{aligned} \lambda_2 &\le \frac{\mathbf{x}^{\mathrm{T}}\mathbf{L}\mathbf{x}}{\mathbf{x}^{\mathrm{T}}\mathbf{D}\mathbf{x}} = \frac{4cut(\mathcal{S}, \bar{\mathcal{S}})}{2C} \\ &= 2\phi(\mathcal{S}) = 2\phi \end{aligned}$$

We then prove the direction of $\frac{\phi^2}{2} \le \lambda_2$. We follow the proof technique and notations used in Theorem 1 of [21]. Let $\mathbf{g}$ denote the solution to (A.1). Using an argument similar to Proposition 2.2, we can show $-1 \le \mathbf{g}(u) \le 1$. We sort the nodes according to $\mathbf{g}$ such that

$$1 = \mathbf{g}(s) = \mathbf{g}(v_1) \ge \cdots \ge \mathbf{g}(v_n) = \mathbf{g}(t) = -1 \tag{A.3}$$

Let $\mathcal{S}_i = \{v_1, \ldots, v_i\}$ for $i = 1, \ldots, n-1$ and we denote by $\widetilde{\mathrm{vol}}(\mathcal{S}_i) = \min\{\mathrm{vol}(\mathcal{S}_i), \mathrm{vol}(\bar{\mathcal{S}}_i)\} = C$. We let $\alpha = \min_{i=1}^{n-1} \phi(\mathcal{S}_i)$. Generalizing the proof to Theorem 1 in [21] we can show

$$\lambda_2 \ge \frac{\left(\sum_{u\sim v} c(\{u,v\})(\mathbf{g}_+(u)^2 - \mathbf{g}_+(v)^2)\right)^2}{\left(\sum_u \mathbf{g}_+(u)^2\mathbf{d}(u)\right)\left(\sum_{u\sim v} c(\{u,v\})(\mathbf{g}_+(u) + \mathbf{g}_+(v))^2\right)}$$

To bound the denominator, we have

$$\begin{aligned} &\sum_{u\sim v} c(\{u,v\})(\mathbf{g}_+(u) + \mathbf{g}_+(v))^2 \\ \le &2\sum_{u\sim v} c(\{u,v\})(\mathbf{g}_+(u)^2 + \mathbf{g}_+(v)^2) \le 2\mathbf{g}_+(s)^2\mathbf{d}(s) \end{aligned}$$

where the last inequality follows from $\mathbf{g}_+(u) \le \mathbf{g}_+(s)$ and $\mathbf{d}(s) = C = 2\sum_{u\sim v} c(\{u,v\})$. Because $\mathbf{g}_+(t) = 0$ and according to the definition of (10) we have $\sum_u \mathbf{g}_+(u)^2\mathbf{d}(u) = \mathbf{g}_+(s)^2\mathbf{d}(s)$. By telescoping the term $\mathbf{g}_+(u)^2 - \mathbf{g}_+(v)^2$ using the ordering (A.3), we then have

$$\begin{aligned} \lambda_2 &\ge \frac{\left(\sum_{i=1}^{n-1}(\mathbf{g}_+(v_i)^2 - \mathbf{g}_+(v_{i+1})^2)cut(\mathcal{S}_i, \bar{\mathcal{S}}_i)\right)^2}{2(\mathbf{g}_+(s)^2\mathbf{d}(s))^2} \\ &\ge \frac{\alpha^2}{2}\frac{\left(\sum_{i=1}^{n-1}(\mathbf{g}_+(v_i)^2 - \mathbf{g}_+(v_{i+1})^2)\widetilde{\mathrm{vol}}(\mathcal{S}_i)\right)^2}{(\mathbf{g}_+(s)^2\mathbf{d}(s))^2} = \frac{\alpha^2}{2} \end{aligned}$$

where the last equality follows from $\sum_{i=1}^{n-1}(\mathbf{g}_+(v_i)^2 - \mathbf{g}_+(v_{i+1})^2)\widetilde{\mathrm{vol}}(\mathcal{S}_i) = \mathbf{g}_+(s)^2\mathbf{d}(s)$ because $\widetilde{\mathrm{vol}}(\mathcal{S}_i) = C = \mathbf{d}(s)$ for $i = 1, \ldots, n-1$. From $\alpha \ge \phi$ we have $\lambda_2 \ge \frac{\phi^2}{2}$. □

References

[1] R. Andersen, K.J. Lang, An algorithm for improving graph partitions, in: S.-H. Teng (Ed.), Proceedings of ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2008, pp. 651–660.
[2] Y. Boykov, G. Funka-Lea, Graph cuts and efficient N-D image segmentation, Int. J. Comput. Vis. 70 (2) (2006) 109–131.
[3] D. Hernando, P. Kellman, J. Haldar, Z. Liang, Robust water/fat separation in the presence of large field inhomogeneities using a graph cut algorithm, Magn. Reson. Med. 63 (1) (2010) 79–90.
[4] P. Christiano, J.A. Kelner, A. Madry, D.A. Spielman, S. Teng, Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs, in: Proceedings of Symposium on Theory of Computing, STOC, ACM, 2011, pp. 273–282.
[5] J.A. Kelner, Y.T. Lee, L. Orecchia, A. Sidford, An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations, in: Proceedings of ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2014, pp. 217–226.
[6] Y.T. Lee, S. Rao, N. Srivastava, A new approach to computing maximum flows using electrical flows., in: Proceedings of Symposium on Theory of Computing, STOC, ACM, 2013, pp. 755–764.
[7] Y. Boykov, V. Kolmogorov, An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision, IEEE Trans. Pattern Anal. Mach. Intell. 26 (9) (2004) 1124–1137.
[8] B.V. Cherkassky, A.V. Goldberg, On implementing the push-relabel method for the maximum flow problem, Algorithmica 19 (4) (1997) 390–410.
[9] A.V. Goldberg, S. Hed, H. Kaplan, R.E. Tarjan, R.F. Werneck, Maximum flows by incremental breadth-first search, in: Proceedings of the 19th European Conference on Algorithms, ESA'11, Springer-Verlag, 2011, pp. 457–468.
[10] D.S. Hochbaum, The pseudoflow algorithm: a new algorithm for the maximum flow problem, Oper. Res. 58 (4) (2008) 992–1009.
[11] J. Nieplocha, A. Marquez, J. Feo, D.G. ChavarrÃa-Miranda, G. Chin, C. Scherrer, N. Beagley, Evaluating the potential of multithreaded platforms for irregular scientific computations, in: U. Banerjee, J. Moreira, M. Dubois, P. Stenstrom (Eds.), Proceedings of International Conference on Computing Frontiers, ACM, 2007, pp. 47–58.
[12] J. Feo, D. Harper, S. Kahan, P. Konecny, Eldorado., in: N. Bagherzadeh, M. Valero, A. Ramirez (Eds.), Proceedings of International Conference on Computing Frontiers, ACM, 2005, pp. 28–34.
[13] A. Tumeo, S. Secchi, O. Villa, Designing next-generation massively multithreaded architectures for irregular applications, IEEE Comput. 45 (8) (2012) 53–61.

[14] A. Morari, A. Tumeo, D.G. Chavarría-Miranda, O. Villa, M. Valero, Scaling irregular applications through data aggregation and software multithreading, in: Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium, IEEE, 2014, pp. 1126–1135.
[15] A. Schrijver, Combinatorial Optimization – Polyhedra and Efficiency, Springer, 2003.
[16] A. Beck, On the convergence of alternating minimization for convex programming with applications to iteratively reweighted least squares and decomposition schemes, SIAM J. Optim. 25 (1) (2015) 185–209.
[17] N. Vishnoi, $\mathbf{Lx} = \mathbf{b}$. Laplacian solvers and their algorithmic applications, Found. Trends Theor. Comput. Sci. 8 (1–2) (2012) 1–141.
[18] M. Benzi, G.H. Golub, J. Liesen, Numerical solution of saddle point problems, Acta Numer. 14 (2005) 1–137.
[19] R.S. Varga, Matrix Iterative Analysis, Second edition, Springer, 2000.
[20] S. Toledo, H. Avron, Combinatorial preconditioners, in: U. Naumann, O. Schenk (Eds.), Combinatorial Scientific Computing, Chapman & Hall/CRC, 2012.
[21] F. Chung, Four Cheeger-type inequalities for graph partitioning algorithms, in: Proceedings of International Conference on Cognitive Modeling, ICCM, 2007.
[22] I. Koutis, G. Miller, R. Peng, A generalized Cheeger inequality, 2014(Personal communication and public presentation).
[23] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, 2003.
[24] G. Karypis, V. Kumar, A parallel algorithm for multilevel graph partitioning and sparse matrix ordering, J. Parallel Distrib. Comput. 48 (1) (1998) 71–95.
[25] G. Karypis, V. Kumar, Parallel multilevel k-way partitioning scheme for irregular graphs, SIAM Rev. 41 (2) (1999) 278–300.
[26] A. Bhusnurmath, C.J. Taylor, Graph cuts via ℓ_1 norm minimization, IEEE Trans. Pattern Anal. Mach. Intell. 30 (10) (2008) 1866–1871.
[27] A.V. Goldberg, Two-level push-relabel algorithm for the maximum flow problem., in: A.V. Goldberg, Y. Zhou (Eds.), Proceedings of Conference on Algorithmic Aspects in Information and Management, AAIM, Volume 5564 of Lecture Notes in Computer Science, Springer, 2009, pp. 212–225.
[28] J.A. Kelner, L. Orecchia, A. Sidford, Z. Zhu, A simple, combinatorial algorithm for solving SDD systems in nearly-linear time, in: Proceedings of the Forty-fifth Annual ACM symposium on Theory of computing, ACM, 2013, pp. 911–920.
[29] I. Koutis, G.L. Miller, D. Tolliver, Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing, Comput. Vis. Image Underst. 115 (12) (2011) 1638–1646.
[30] O.E. Livne, A. Brandt, Lean Algebraic Multigrid (LAMG): fast graph Laplacian solver, SIAM J. Sci. Comput. 34 (4) (2012) B499–B522.
[31] K. Lang, Fixing two weaknesses of the spectral method., in: Proceedings of Neural Information Processing Systems, NIPS, 2005.
[32] V. Kolmogorov, R. Zabih, What energy functions can be minimized via graph cuts? IEEE Trans. Pattern Anal. Mach. Intell. 26 (2) (2004) 147–159.
[33] A. Shekhovtsov, V. Hlavac, A distributed mincut/maxflow algorithm combining path augmentation and push-relabel, Int. J. Comput. Vis. 104 (3) (2013) 315–342.
[34] A. Delong, Y. Boykov, A scalable graph-cut algorithm for N-D grids, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, IEEE Computer Society, 2008.
[35] P. Strandmark, F. Kahl, Parallel and distributed graph cuts by dual decomposition, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2010, pp. 2085–2092.
[36] T.A. Davis, Y. Hu, The University of Florida sparse matrix collection, ACM Trans. Math. Softw. 38 (1) (2011) 1.
[37] C. Russell, L. Ladicky, P. Kohli, P.H.S. Torr, Exact and approximate inference in associative hierarchical networks using graph cuts, in: P. Grunwald, P. Spirtes (Eds.), Proceedings of Conference on Uncertainty in Artificial Intelligence, UAI, AUAI Press, 2010, pp. 501–508.
[38] D.S. Johnson, C.C. McGeoch (Eds.), Network Flows and Matching: First DIMACS Implementation Challenge, American Mathematical Society, 1993.
[39] B. Smith, PETSc (portable, extensible toolkit for scientific computation), in: D.A. Padua (Ed.), Encyclopedia of Parallel Computing, Springer, 2011, pp. 1530–1539.
[40] T.A. Davis, Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method, ACM Trans. Math. Softw. 30 (2) (2004) 196–199.