基于 FPGA 的稀疏网络关键节点计算的硬件加速方法研究

史圣卿*① 陈 凯② 汪 玉① 罗 嵘①
①(清华大学电子工程系清华信息科学与技术国家实验室(筹) 北京 100084)
②(海南省通信管理局 海口 570206)

摘 要：随着互联网、生物医学及社交网络等复杂网络研究的深入，如何寻找其等效图中关键节点越来越重要。中介中心度作为衡量图中节点重要性的主要指标，其单点的计算复杂度高达 $O(N^3)$，因而成为关键节点计算问题的难点。该文在对传统的中介中心度快速算法进行分析之后，提出了一种适用于硬件设计的改进算法。同时，基于算法中各点独立、以及相邻计算间无数据依赖的特点，该文利用改进算法实现了一个流水线结构的 8 计算单元并行计算系统，并在 FPGA 上完成了硬件系统的设计和验证。通过对比 8 核 CPU 软件系统的计算时间，该文的硬件计算系统实现了 4.31 倍的加速比。

关键词：FPGA；中介中心度；硬件计算；复杂网络；图
中图分类号：TN402 **文献标识码**： A **文章编号**：1009-5896(2011)10-2536-05
DOI: 10.3724/SP.J.1146.2011.00363

Node Importance Analysis in Complex Networks Based on Hardware Computing

Shi Sheng-qing① Chen Kai② Wang Yu① Luo Rong①
①(*TNList*, *Dept. of Electronic Engineering*, *Tsinghua University*, *Beijing* 100084, *China*)
②(*Hainan Communications Administration*, *Haikou* 570206, *China*)

Abstract: Betweenness centrality is a widely used indicator to measure the node importance in complex network s, but it is computationally-expensive to calculate betweenness centrality. In this paper, analysis on the traditional betweenness centrality algorithms is completed and a novel algorithm is proposed to meet the hardware design features. Based on this algorithm, parallel computing system is implemented on FPGA with task level coarse grained parallelism and pipeline based fine grained parallelism. The experimental results show that the FPGA based implementation achieves up to 4.31 times speedup compared with an 8-core CPU implementation.

Key words: FPGA; Betweenness centrality; Hardware computing; Complex networks; Graphs

1 引言

图是一种基本的数据形式，大量现实世界中的复杂网络如互联网、道路交通、生物医学等问题都需要抽象成稀疏图来表达，并通过图的相关属性来进行分析。传统的研究主要集中在计算图中的最短距离[1,2]。近年来，随着基于核磁共振成像的生物医学网络[3]以及基于人际关系的社交网络等研究的不断深入[4]，使得人们对于图属性的计算需求越来越大，关注的重点也由寻找最短距离转移到寻找图中的关键节点。

中介中心度(betweenness centrality)[5]是衡量图中节点重要性的主要指标。相对于单点最短距离的计算复杂度 $O(N^2)$，单点中介中心度的计算复杂度 $O(N^3)$要高出一个数量级[6]。因此计算时间成为限制关键节点研究的主要问题，现有的对中介中心度计算进行加速的工作主要都是基于多核 CPU[3]和超级计算机[3]。基于分布式存储的超级计算机如 Cray MTA-2[1]可以实现较大的网络规模和较高的计算速度，但是其造价高昂，功耗也很大，难以普及；多核 CPU 计算中介中心度主要采用任务级并行的方式，难以实现更细粒度的并行，而受限于存储访问瓶颈[7]，其任务级并行的效果相比于单核 CPU 提升也较为有限。相比之下，FPGA 在成本、功耗和运算速度等方面均具有一定的优势[8]。

本文在对传统的中介中心度快速算法进行分析之后，首先针对 FPGA 硬件存储结构较为固定的特点，提出了一种不需要指针点集的改进型中介中心度算法以适应硬件设计；然后针对算法中各源点的计算完全独立以及计算过程中邻近节点间无数据依

2011-04-14 收到，2011-06-23 改回
*通信作者：史圣卿 shisq04@mails.tsinghua.edu.cn

赖的特点，在改进型算法的基础上，搭建了一个 8 计算单元的硬件计算系统实现任务级的粗粒度并行，同时各计算单元采用全流水线结构实现细粒度并行。本文在 FPGA 上完成了硬件系统的设计与验证，利用该硬件系统计算一个 4000 节点稀疏图的中介中心度，比利用 8 核 CPU 的计算时间缩短了 4.31 倍。

本文内容安排如下：第 2 节为中介中心度算法介绍，第 3 节提出了针对硬件设计的改进型算法，第 4 节介绍了基于 FPGA 的硬件系统架构和计算单元设计，并对比了软硬件的计算时间，最后是总结。

2　中介中心度算法介绍

节点的中心度是用来衡量节点在图中关键程度的重要指标。常见的中心度指标包括度中心度(degree centrality)，邻近中心度(closeness centrality)和中介中心度(betweenness centrality)等[5]。与前两项指标相比，中介中心度是基于最短路径得到，因此中介中心度衡量的是节点在图的拓扑结构中的重要性，其参考价值最大。

2.1 中介中心度的数学描述

图问题通常抽象描述为 $G=(V, E)$，其中 V 是图中所有点的集合，E 是图中所有边的集合。令 n 和 m 分别表示图中的总节点数和总边数。本文所研究的网络问题中，假定不存在孤立的点集，并且所有边的长度 $\omega(e)$ 都是正实数($\omega(e)>0$)，则对于图中任意两个节点 s 和 $t(s, t\in V)$，必存在连接两点的最短距离 $d(s,t)$以及对应的最短路径。定义 σ_{st} 为两点间的最短路径条数，因为两点间的最短路径一定存在且可能有多条，所以 $\sigma_{st}\in[1,m)$。对于节点自身，$d(s,s)=0$，$\sigma_{ss}=1$。再定义 $\sigma_{st}(v)$，表示点 s 到点 t 的最短路径中通过点 v 的条数。利用 $\sigma_{st}(v)$和 σ_{st} 定义中介中心度 $C_B(v)$如式(1)：

$$C_B(v)=\sum_{s\neq v\neq t\in V}\frac{\sigma_{st}(v)}{\sigma_{st}} \tag{1}$$

由式(1)可以看出，$C_B(v)$代表了图中所有节点之间的最短路径通过点 v 的比重。因此中介中心度直观地衡量了点 v 在图的拓扑结构中的重要程度。

2.2 中介中心度的快速算法描述

Brandes[9]在 2001 年提出了一种计算中介中心度的快速算法，此后大部分关于中介中心度的研究都是基于该快速算法[4,6,7]。

Brandes 提出的快速算法中，首先定义 $\sigma_{st}(v)$ 和 $\delta_s(v)$，分别表示点 s 到点 t 的最短路径通过点 v 的比重，以及点 s 到全图各点的最短路径通过点 v 的比重。$\sigma_{st}(v)$和 $\delta_s(v)$ 可以解释为点 s 到点 t 对点 v 的依赖度，以及点 s 到全图对点 v 的依赖度。

$$\delta_s(v)=\sum_{t\in V}\delta_{st}(v)=\sum_{t\in V}\frac{\sigma_{st}(v)}{\sigma_{st}} \tag{2}$$

再定义点 v 相对源点 s 的前驱点集 $P_s(v)$, $P_s(v)=\{u\in V:\{u,v\}\in E, d(s,v)=d(s,u)+\omega(u,v)\}$，表示 v 的所有邻居节点中，位于从 s 到 v 最短路径上的那些邻居点。定义了 v 的前驱点集 $P_s(v)$之后，可以将源点 s 对 v 的依赖度 $\delta_s(v)$ 分解为所有以 v 为前驱点的点 w_i对 $\delta_s(v)$ 的贡献之和。考虑到 w 可能有多个前驱节点，则 $\delta_s(w)$ 应该按照 w 经过各个前驱点到 s 的最短路径数量来进行分配，路径数量越多，w 的依赖度越多的来自于该前驱点。由 $\delta_s(w)$ 的贡献计算 $\delta_s(v)$ 的过程可如图 1 所示，通过分析可以得到，w 对 $\delta_s(v)$ 的贡献应该等于 $\delta_s(w)$ 乘以比例系数 σ_{sv}/σ_{sw}。

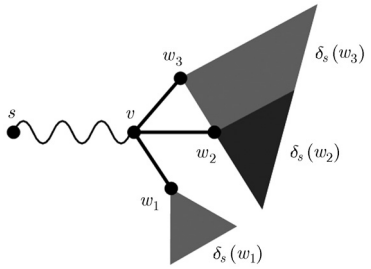

图 1　$\delta_s(v)$ 的计算

根据式(1)和式(2)，将图中所有点 s 对点 v 的依赖度累加起来，即为点 v 在图中的中介中心度，因此中介中心度的计算公式可如式(3)所示。

$$C_B(v)=\sum_{s\in V}\delta_s(v)=\sum_{s\in V}\sum_{w:v\in P_s(w)}\frac{\sigma_{sv}}{\sigma_{sw}}(1+\delta_s(w)) \tag{3}$$

根据式(3)，中介中心度 $C_B(V)$的计算可以分为两步，第 1 步是对于给定的源点 s，计算 s 对 v 的依赖度 $\delta_s(v)$，第 2 步是将所有源点 s 对 v 的依赖度累加起来。由图 1 可以知道，对于给定的源点 s，如果图中的每一个点都只保留与前驱点连接的边而删除其他的边，则会最终构成一个以源点 s 为根节点的广度优先生成树，这个生成树的所有叶子节点是只有自己的一条最短路径通过的，即 $\delta_s(v)=1$，生成树上其他节点 v 的 $\delta_s(v)$ 则由他的子节点累加得到。因此，由 s 的广度优先树从下向上逐层累加可以得到图中所有点 v 的 $\delta_s(v)$。

2.3 中介中心度快速算法的软件实现

根据 2.2 节中的分析，中介中心度快速算法的核心思路可以描述为对图中的每一个点 s 进行广度优先搜索，并根据生成树得到点 s 对图中其他所有点 v 的贡献 $\delta_s(v)$。完成对所有源点 s 的计算之后，再汇总各点的 $\delta_s(v)$ 即可得到所有点 v 的中介中心度 $C_B(v)$。

对于单个源点 s 的计算过程，则分为前向计算和后向计算两部分进行。

(1)前向计算：以 s 为源点进行广度优先搜索，搜索过程中，每当有节点的距离被更新，便将其添加至队列 Q 及栈 S，同时更新节点的最短距离，并保存节点的前驱点以及节点到源点的路径数量。

(2)后向计算：从栈中依次取出节点，即为前向搜索的相反顺序。逐点向上累加求得 s 对各点 v 的依赖度 $\delta_s(v)$，并累加至 $C_B(v)$。

算法如表 1 所示，时间复杂度为 $O(nm+n^2\log n)$，空间复杂度为 $O(n+m)$。表 1 中第(5)行–(15)行对应单个源点的前向计算阶段，(16)行–(22)行对应后向计算阶段。

表 1 计算中介中心度的伪代码

```
(1)  初始化：令所有 C_B[v]=0
(2)  对图中每个点 s 执行一次：
(3)    清空 Q, S, P[w]；令 d[s]=0, σ[s]=1
(4)    将源点 s 加入队列 Q
(5)    前向计算：
(6)      若 Q 非空，从 Q 中取出点 v，推入栈 S
(7)        对 v 的所有邻居 w：
(8)          若 d[w]=d[v]+d[v,w]
(9)            σ[w] = σ[w] + σ[v]
(10)           将 v 加入 P [w]
(11)         若 d[w]<0 或 d[w]>d[v] +d[v,w]
(12)           d[w]=d[v]+d[v,w]
(13)           σ[w]= σ[v] , P[w] = v
(14)           若 w 不在队列 Q 中
(15)           将 w 加入 Q
(16)   后向计算：
(17)   当 S 非空时，从 S 中取出点 w
(18)     若 w 是第 1 次出栈：
(19)       对 P [w]中的所有点 v
(20)       δ[v] = δ[v] + σ[v] / σ[w](1 + δ[w])
(21)     若 w≠s，累加计算 C_B[w]
(22)       C_B [w]=C_B [w]+ δ[w]
```

3 针对硬件设计的算法改进

基于 FPGA 进行硬件设计时，系统结构和片上计算资源的分配方式十分灵活，但是存储结构的设计和片上存储空间都较为受限，在软件上大量使用的指针等存储结构在硬件上实现困难。而传统的中介中心度算法以及一些并行算法[6]中，各点的前驱点集占据了 $O(m)$的存储空间，并且各点的前驱点数不固定，使得前驱点集的存储空间难以预先分配。在软件上可以利用指针结构来进行存储解决这个问题，但在硬件实现上则存在较大的困难。

针对这个问题，本文提出了一种改进型中介中心度算法。考虑到在有权图的条件下，两条不同的路径距离完全相同的情况较难出现，可以认为每两点之间有且只有一条最短路径存在。因此，对于图中的任意点 v，一定只有一个前驱节点存在。由此对表 1 中的算法做如下改进：

(1)对每一个点，不再保留前驱点集，而仅保留一个前驱点，每次更新最短距离的同时更新该前驱点。节省前驱点集所占据的大量存储空间，并省略掉添加前驱点集的操作。

(2)不再统计路径数 $\sigma[v]$，因为全部假定为 1。同时在后向计算过程中，各节点累计的依赖度 $\delta[v]$ 不再需要分配，而是直接累加到它的前驱节点。

改进后算法的总体结构不变，仍然是对所有点 s 做计算并累加，与表 1 算法的区别在于前向计算和后向计算的具体操作。改进后的前向计算和后向计算过程如表 2 所述。经过改进后，前驱点集的空间被大幅压缩，空间复杂度由 $O(n+m)$减为 $O(n)$。时间复杂度仍为 $O(nm+n^2\log n)$，但因为省去了添加前驱点集的操作，计算时间也有所减少。表 2 中第(6)行和第(13)行标识出了算法的改进，对比原算法为表 1 中第(10)行和第(20)行。

表 2 针对硬件设计的算法改进

```
(1)  前向计算：
(2)    若 Q 非空，从 Q 中取出点 v，推入栈 S
(3)      对 v 的所有邻居 w：
(4)        若 d[w]<0 或 d[w]>d[v] + d[v,w]
(5)          d[w]=d[v] + d[v,w]
(6)          P[w]=v
(7)        若 w 不在队列 Q 中
(8)          将 w 加入 Q
(9)  后向计算：
(10)   若 S 非空，从 S 中取出点 w
(11)     若 w 是第 1 次出栈：
(12)       对 v=P[w]
(13)         δ[v] = δ[v] + 1 + δ[w]
(14)       若 w≠s，累加计算 C_B[w]
(15)         C_B [w]=C_B[w]+ δ[w]
```

通过软件仿真对比采用改进算法的计算结果与实际值的误差，如图 2 所示。测试所用网络为 4000 个点，60000 条边的随机生成图。将结果按照中介中心度的实际值大小排序，并截取中介中心度最大的前 1000 个节点。图 2(a)为前 1000 个节点中介中心度的实际值，图 2(b)为前 1000 点的改进算法计算结果以及相对实际值的误差。通过对比可见，改进算法得到的中介中心度结果与实际值误差在 10%以

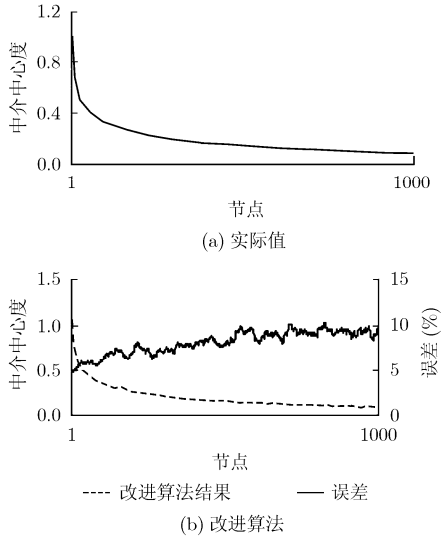

图 2 改进算法与实际值的结果对比

内，并且对于节点中介中心度的排序没有影响，因此利用改进算法进行关键节点分析是完全可行的。

4 基于 FPGA 的硬件系统设计

对于一个硬件计算系统，需要在合适的层面上实现并行化以达到加速的效果。在前述中介中心度算法中，存在两个可以进行并行计算的层面：

(1)对不同源点 s 进行前向计算和后向计算求 $\delta_s[v]$ 的过程是完全独立的，可以同时对多个源点 s 进行该计算过程。

(2)前向计算过程中，从队列 Q 中取出一个节点 v 后，对 v 的所有邻居节点计算距离和比较的过程是完全独立的，可以同时对这些节点进行更新。

针对第 1 个并行层面，本文构建了一个多计算单元并行的系统架构。而对于第 2 个并行层面，本文在计算单元中采用了流水线结构设计。下面分别从系统总体架构和计算单元两部分来介绍硬件系统设计。

4.1 系统总体架构

系统由多个计算单元并行组成，每一个计算单元独立完成一个节点的计算。在节点的计算过程中，有一部分信息是由各个计算单元单独保存的，包括图中各点 v 对于源点 s 的 $d_s[v]$, $\delta_s[v]$；除此之外，图本身的信息包括各点的邻居节点编号以及到邻居节点的距离(即对应的边长)是始终固定不变的，这些信息会保存在统一的存储空间上。

系统总体架构如图 3 所示，多个计算单元全部连接在存储器控制单元上，图信息则存储在一块单独的 RAM 中。在计算过程中图信息是只读的，通过存储器控制单元控制计算单元对图信息的访问。

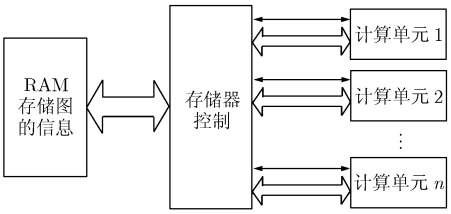

图 3 多计算单元的系统总体架构

图信息所需的存储空间规模为 $O(m+n)$，为了节约存储空间，本文采用邻接数据结构 CSR (compressed sparse row)对图信息进行存储。对于大规模的网络，可以将图信息存储在片外的存储器上，并通过存储器控制单元与其进行数据传输。计算单元内部所需的存储空间规模为 $O(n)$，这一部分资源必须在片上生成。

4.2 流水线结构计算单元

计算单元的设计基于表 2 所示的改进型算法，利用队列 Q 和栈 S 分别管理前向计算和后向计算的节点顺序。各点 v 至源点 s 的最短距离 $d_s[v]$，前驱点 $P[v]$以及对中介中心度的贡献 $\delta_s[v]$，这些都通过片上 RAM 在计算单元内部实现。计算单元的硬件结构如图 4 所示，其运算过程如下：

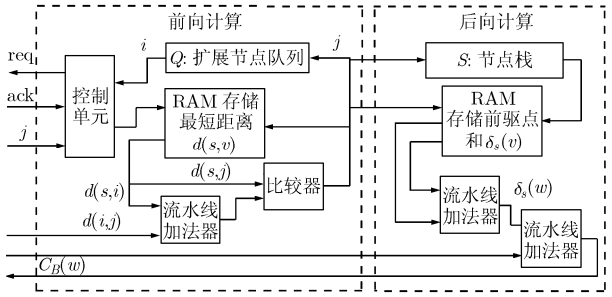

图 4 计算单元结构图

(1)前向计算时，控制单元连续从队列 Q 中读出新的节点，并向外部存储器请求节点的邻居及边长信息。

(2)将边长及邻居点的当前距离送至流水线加法器，与节点距离比较，若节点距离被更新，则将邻居点加入队列 Q 和栈 S，同时更新前驱点。

(3) 后向计算时，不断从栈中取出节点，将节点及前驱点的 δ 值送至流水线加法器，并将 δ 与 C_B 累加后更新至 C_B。

通过控制单元保证连续读取队列 Q 和栈 S，可以保证计算单元中的数据流水，提高计算效率。

4.3 实验方法及实验结果

基于上述硬件结构，本文实现了具有 8 个计算单元的硬件计算系统，其中每一个计算单元对应的

节点规模为 4096 点。为了测试硬件系统的计算效果，在 MATLAB 中生成了 3 组测试数据，分别为 1000 点，2000 点和 4000 点的稀疏图，3 张图的平均节点度数均为 16，边长的取值范围为 0～10000。

本文所用的软件计算平台为一台包含两块 4 核 CPU(Xeon E5405 2 GHz) 以及 8 GB DDR2 SDRAM 内存的服务器，记录了 8 核 CPU 的软件计算时间。硬件系统实现所选用的 FPGA 芯片型号为 Altera Stratix II EP2S180F1508I4，在 Quartus II 9.1 中对系统进行编译并利用 Modelsim 6.5 进行硬件仿真。经过对硬件系统关键路径的时序分析，系统的最高工作频率为 113.4 MHz。将时钟频率设置为 100 MHz，对硬件系统的计算时间进行了仿真。硬件和软件的计算时间如表 3 所示。

表 3 FPGA 与 8 核 CPU 计算时间对比

节点数	1000	2000	4000
CPU 计算时间(ms)	122	420	1638
FPGA 计算时间(ms)	25.6	101.9	380.1
加速比	4.77	4.12	4.31

通过对比，在网络规模为 4000 点时，利用硬件计算中介中心度可以达到 4.31 倍的加速比。

4.4 扩展性分析

4.3 节中实现的 4096 点 8 计算单元的硬件计算系统，计算单元所占用片上存储空间达到 5.7 Mbit，已占所选芯片总存储资源的 85%。如 4.1 节所述，计算单元内部所使用的存储很难移至片外，因此系统的规模主要受限于 FPGA 的片上存储资源。

为了证实系统的扩展性，本文将计算单元的节点规模扩展为 8192 点，仍然选用 Altera Stratix II EP2S180F1508I4 芯片。受片上存储资源的限制，仅实现 4 个计算单元。4096 点 8 计算单元和 8192 点 4 计算单元两种结构的资源占用如表 4 所示，两种结构分别需要占据片上 85%和 89%的块存储资源，而片上的组合逻辑单元和寄存器资源则很充裕。

为了测试 8192 点 4 计算单元结构的计算效果，生成一组 8000 点的随机图作为测试数据，同样在 100 MHz 的时钟频率下仿真，得到 4 个计算单元的硬件系统完成 8000 点中介中心度计算的时间为 2629.16 ms，对比的 8 核 CPU 计算时间为 5609 ms。在计算单元减半的情况下，实现加速比 2.13 倍。

表 4 FPGA 片上资源占用情况

节点规模	4096		8192		总资源
计算单元个数	8		4		
组合逻辑单元	12,395	12%	4,545	4%	106,032
寄存器	5,588	5%	3,522	3%	106,032
块存储(kbit)	5,718	85%	6,002	89%	6,748

5 结束语

关键节点计算是图分析中的研究重点，本文针对衡量节点重要性的中介中心度参数，改进了软件算法使其适合硬件设计，并基于 FPGA 实现了整个硬件计算系统。任务级并行的多计算单元结构和基于流水线的计算单元设计发挥了硬件计算的优势。最终实现了相对于 8 核 CPU 系统 4.31 倍的加速比。

参 考 文 献

[1] Bader D A and Madduri K. Designing multithreaded algorithms for breadth-first search and *st*-connectivity on the Cray MTA-2. International Conference on Parallel Processing (ICPP'06), Columbus, OH, USA, August 14–18, 2006: 523–530.

[2] Luo Li-juan, Wong Martin, and Hwu Wen-mei. An effective GPU implementation of breadth-first search. Proceedings of the 47th Design Automation Conference(DAC'10), Anaheim, CA, USA, June 13–18, 2010: 52–55.

[3] Bader D A and Madduri K. A graph-theoretic analysis of the human protein-interaction network using multicore parallel algorithms. IEEE International Parallel and Distributed Processing Symposium (IPDPS'07), Long Beach, CA, USA, March 26–30, 2007: 1–8.

[4] Ediger D, Jiang K, Riedy J, *et al*.. Massive social network analysis: mining twitter for social good. 39th International Conference on Parallel Processing (ICPP'10), San Diego, CA, USA, Sept. 13–16, 2010: 583–593.

[5] Freeman L C. A set of measures of centrality based on betweenness. *Sociometry*, 1977, 40(1): 35–41.

[6] Madduri K, Ediger D, Jiang K, *et al*.. A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets. IEEE International Symposium on Parallel and Distributed Processing (IPDPS'09), Rome, Italy, May 23–29, 2009: 1–8.

[7] Tan Guang-ming, Tu Deng-biao, and Sun Ning-hui. A parallel algorithm for computing betweenness centrality. International Conference on Parallel Processing (ICPP' 09), Vienna, Austria, Sept. 22–25, 2009: 340–347.

[8] Bondhugula U, Devulapalli A, Fernando J, *et al*.. Parallel FPGA-based all-pairs shortest-paths in a directed graph. 20th International Parallel and Distributed Processing Symposium (IPDPS'06), Rhodes Island, Greece, Apr. 25–29, 2006: 90–99.

[9] Brandes U. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 2001, 25(2): 163–177.

史圣卿： 男，1987 年生，硕士生，研究方向为大规模集成电路设计、基于 FPGA 的硬件计算等.

陈 凯： 男，1979 年生，硕士，研究方向为网络优化、FPGA 设计等.

汪 玉： 男，1982 年生，博士，讲师，研究方向为大规模集成电路设计技术等.

罗 嵘： 女，1970 年生，博士，副教授，研究方向为大规模集成电路设计技术、多媒体处理系统设计技术等.