Advanced Computer Architecture:

A Google Search Engine

Jeremy Bradley

`Room 372. Office hour - Thursdays at 3pm. Email: jb@doc.ic.ac.uk`

`Course notes: http://www.doc.ic.ac.uk/∼jb/`

Department of Computing, Imperial College London

Produced with prosper and LaTeX

Introduction to PageRank

- PageRank is used by Google to order pages which have the same search terms
- Documented by Google founders: Sergey Brin and Lawrence Page
  - "The PageRank Citation Ranking: Bringing Order to the Web" Page, Brin, Motwani and Winograd
  - "The Anatomy of a Large-Scale Hypertextual Web Search Engine" Brin and Page
  - "Extrapolation Methods for Accelerating PageRank Computations" Kamvar, Haveliwala, Manning and Golub

Motivation for PageRank

- PageRank introduced to solve the junk web-page problem
- By 1997, even a specific search query would generate 100s of results
- November 1997: "...only one of the top four commercial search engines finds itself"! i.e. places itself in its own top ten search results

Search Result Manipulation (I)

- Search engines ordered results returned for the same query terms according to:
  - page content
  - URL
  - page title
  - user presented meta data
  - frequency of occurance of search term/related terms
- This is all user controllable data

$\Rightarrow$ Web authors could manipulate it to enhance their search ordering

Search Result Manipulation (II)

- Web pages that wanted to popularise themselves:
  - put repeated dummy search terms into web pages to catch search engine traffic
  - competitor web pages (even reputable ones) had to do likewise
  - web pages ballooned in size from junk content
- user controllable page content quickly became no judge of page quality or relevance

Solution: PageRank

- PageRank designed to overcome problem
  - based on research-style citations
- A page is considered more useful if:
  - many pages refer to (link) to it
  - small number of important pages refer to it
- A page is considered less useful:
  - if few or no pages link to it
- PageRank is independent of the page content
  - i.e. importantly does not have to be recalculated for each query

What is PageRank

- PageRank is based on underlying web graph
  - measure of page interconnectedness
- For a given web page, its PageRank is:
  - proportional to the number of pages that link to it
  - is a value between 0 and 1
  - propagated recursively to all the pages that the page links to
  - does not bear any "linear" relationship to the quoted PageRank figure (between 0 and 10) that you get from the Google toolbar in Windows

PageRank's Shortcomings

- the accumulated PageRank for a site is much harder to manipulate BUT...
- dependent on link-structure i.e. links not being broken
- works well over static web structure but poorly over dynamic or query-driven structure
- susceptible to *Google spam*
  - i.e. large communities of people collaborating to link to each others pages

Derivation of PageRank

- Consider G the underlying web graph.
  - The nodes of G are web pages
  - A directed edge from page u to page v represents a hypertext link on u which points to v; written $u \longrightarrow v$
- Construct transition matrix P from graph G by letting $P_{ij} = 1/\,\text{deg}(u_i)$ if there is a link $u_i \longrightarrow u_j$ in G and 0 otherwise.
- Is this uniform distribution a fair assumption?

The Random Surfer

$$e.g.\ P = \begin{pmatrix} \vdots & \vdots & \vdots & & \vdots & \vdots \\ 1/3 & 1/3 & 0 & \cdots & 0 & 1/3 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \end{pmatrix}$$

- Example row shows a page linking to 3 other pages u_1, u_2 and u_n
- What happens if a page has no out-links?
  - Get an all-zero row
- Matrix represents a *random surfer* who, with equal probability, follows any of the links that they find on a page

A Markov Chain

- P can also be viewed as a transition matrix of a discrete-time Markov chain
- The PageRank vector represents the steady-state vector of the Markov chain
  - i.e. the probability that the random surfer goes to a particular page after a large number of transitions
- However the pages with no out-links will terminate the surfing (are absorbing states) and distort the steady-state solution

Treating cul-de-sac Pages

- To solve absorbing page problem – if surfer ends up in a page with no out-links:
  - assign probability that surfer will go to any other page (e.g. via bookmarks or typing in a URL) according to personal vector, $\vec{p}$

$\Rightarrow$ replace all zero rows in P with $\vec{p}$

  - $P' = P + D$ where $D = \vec{d}\vec{p}^T$

$$d_i = \begin{cases} 1 & : \text{ if } \deg(u_i) = 0 \\ 0 & : \text{otherwise} \end{cases}$$

Personalisation Vector

- Assumption that $\vec{p}$ taken as: $\begin{pmatrix} \frac{1}{n} \\ \vdots \\ \frac{1}{n} \end{pmatrix}$

- Outer product: $(\vec{d}\vec{p}^T)_{ij} = \sum_{k=1}^{1} d_{ik} p_{kj} = d_i p_j$

$\Rightarrow$ e.g. $D = \begin{pmatrix} \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \\ \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}$

Teleportation Matrix

- Have not yet represented surfer that ignores links on a given page and randomly goes to another (unlinked) page anyway
- This behaviour is given by the *teleportation matrix*, E
- Now: $A = cP' + (1-c)E$ where $E = \tilde{\mathbf{1}}\vec{p}^T$
- i.e. $E = \begin{pmatrix} \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \\ \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \end{pmatrix}$ for $\tilde{\mathbf{1}} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$

Teleportation

- In equation $A = cP' + (1-c)E$
  - $c = \mathbb{P}(\text{link/redirection on page is taken})$
  - $(1-c) = \mathbb{P}(\text{random page is visited})$
  - $c \approx 0.85$
- In Markov chain terms:
  - Prevents process getting *livelocked* in cliques of states
  - Process with transition matrix A is now irreducible (can reach any state from any other state)

PageRank Solution

- PageRank represented by iterative technique, *Power method*:

$$\vec{x}_{(k+1)} = \vec{x}_{(k)} A$$

- Until convergence is achieved
- Need to solve equation:

$$\vec{\pi} = \vec{\pi} A$$

where $\vec{\pi} = \lim_{k \to \infty} \vec{x}_{(k)} = \lim_{k \to \infty} \vec{x}_{(0)} A^k$

PageRank Solution (II)

$$\vec{\pi} = \lim_{k \to \infty} \vec{x}_{(0)} A^k$$

- PageRank algorithm depends crucially on the sparsity of the original matrix P:
  - to keep the matrix–vector multiplication efficient
  - to ensure quick convergence of algorithm
- For a sparse system matrix–vector multiplication can be $O(n)$ rather than $O(n^2)$
- Even for a web graph of 3 billion nodes, convergence can be achieved within about 80 iterations

PageRank Algorithm

- Basic operation: $\vec{x}_{(k+1)} = \vec{x}_{(k)} A$
- A is dense matrix – so need to transform this operation into a sparse matrix calculation involving P
- Trying to show that:

$$\vec{x}_{(k+1)} = c\vec{x}_{(k)} P + (||\vec{x}_{(k)}||_1 - c||\vec{x}_{(k)} P||_1)\vec{p}^T$$

- Need definition of 1-norm of a vector:

$$||\vec{a}||_1 = \sum_i |a_i|$$

PageRank Algorithm I

$$\begin{aligned}\vec{x}_{(k+1)} &= c\vec{x}_{(k)}P' + (1-c)\vec{x}_{(k)}E \\ &= c\vec{x}_{(k)}P + c\vec{x}_{(k)}D + (1-c)\underbrace{\vec{x}_{(k)}\tilde{\mathbf{1}}}_{=||\vec{x}_{(k)}||_1}\vec{p}^T\end{aligned}$$

- Now look at $c\vec{x}_{(k)}D$ term:

$$\begin{aligned}c\vec{x}_{(k)}D &= c(\vec{x}_{(k)}\vec{d})\vec{p}^T \\ &= c\left(\sum_i I_{\{\text{deg}(u_i)=0\}}x_i\right)\vec{p}^T \\ &= c\left(||\vec{x}_{(k)}||_1 - \sum_i I_{\{\text{deg}(u_i)>0\}}x_i\right)\vec{p}^T\end{aligned}$$

PageRank Algorithm II

- Consider term $\vec{x}_{(k)}P = \sum_{j=1}^{n} x_j p_{ji}$

$$
\begin{aligned}
||\vec{x}_{(k)}P||_1 &= \sum_{i=1}^{n}\sum_{j=1}^{n} x_j p_{ji} \\
&= \sum_{j=1}^{n} x_j \sum_{i=1}^{n} p_{ji} \\
&= \sum_{j=1}^{n} x_j \text{ . sum of prob. in row } j \text{ of } P \\
&= \sum_{j=1}^{n} x_j I_{\{\text{deg}\,(u_j)>0\}}
\end{aligned}
$$

PageRank Algorithm III

- Now $c\vec{x}_{(k)}D = c(||\vec{x}_{(k)}||_1 - ||\vec{x}_{(k)}P||_1)\vec{p}^T$
- Back to $(k+1)$th iterate, $\vec{x}_{(k+1)}$:

$$
\begin{aligned}
&= c\vec{x}_{(k)}P + c\vec{x}_{(k)}D + (1-c)||\vec{x}_{(k)}||_1\vec{p}^T \\
&= c\vec{x}_{(k)}P + (||\vec{x}_{(k)}||_1 - c||\vec{x}_{(k)}P||_1)\vec{p}^T
\end{aligned}
$$

- Proof by induction on k for $\vec{x}_{(k+1)} = \vec{x}_{(k)}A$ that $||\vec{x}_{(k)}||_1 = 1$ for all k, so:

$$\vec{x}_{(k+1)} = c\vec{x}_{(k)}P + (1 - c||\vec{x}_{(k)}P||_1)\vec{p}^T$$

PageRank Algorithm IV

- Gives rise to quoted algorithm:
  1. Start with $\vec{x}_{(0)} =$ any vector
  2. Let $\vec{y} = c\vec{x}_{(k)}P$
  3. Set $\omega = ||\vec{x}_{(k)}||_1 - ||\vec{y}||_1$
  4. Next iterate: $\vec{x}_{(k+1)} = \vec{y} + \omega\vec{p}^T$
  5. Repeat from 2. until $||\vec{x}_{(k+1)} - \vec{x}_{(k)}||_1 < \epsilon$
- Why not $\omega = 1 - ||\vec{y}||_1$?
- What's the complexity of this?
- How does it improve over direct $\vec{x}_{(k+1)} = \vec{x}_{(k)}A$ approach?

PageRank Analysis

- Complexity/operation count
  1. $\vec{y} = c\vec{x}_{(k)}P$: sparse multiplication $\Rightarrow O(n)$
  2. $\omega = ||\vec{x}_{(k)}||_1 - ||\vec{y}||_1$: 1-norm of one (or two) $1 \times n$ vectors $\Rightarrow O(n)$
  3. $\omega\vec{p}^T$: scalar multiplication of $1 \times n$ vector $\Rightarrow O(n)$
  4. $\vec{x}_{(k+1)} = \vec{y} + \omega\vec{p}^T$: addition of two $1 \times n$ vectors $\Rightarrow O(n)$
  5. $||\vec{x}_{(k+1)} - \vec{x}_{(k)}||_1 < \epsilon$: vector subtraction and 1-norm $\Rightarrow O(n)$

Teleporting Probability

The effect of changing the parameter, c:

- If $c \leq 0.85$: convergence is fast
- As $c \longrightarrow 1$: convergence is slowed
- However, if c is decreased too far:
  - Google spam becomes more of a problem. i.e. clusters of interlinked pages that are trying to gain high PageRank have a higher probability of being visited at random

PageRank Assumptions

- Uniform distribution of choice of link on a given page
- Personalisation vector $\vec{p}$ assumes uniform distribution across all web pages
- The same personalisation vector is used at page cul-de-sacs as well as in teleportation
- Probability of teleporting, $1 - c$, at a given page is the same at each page

Google Enhancements I

- User classes
  - Different categories of user might have different values of $\vec{p}$ and c
  - Requires a separate PageRank calculation for each user class
  - With for example 10 user classes:

  $\Rightarrow$ 800 iterations of 3 billion by 3 billion matrix in 4 weeks

  $\Rightarrow$ 37,000 matrix calculations per second per computer across 2000 computers (assuming 15 links per page)

Google Enhancements II

- Ideally have a *user class* per individual but not scalable
- Base calculation of $\vec{p}$, c on:
  - Observed link-following behaviour from a Google search
  - Cookie analysis (set expiry date to 2039!)
  - Google toolbar (record every URL visited?)

Implementation on a Cluster

- Vector addition, subtraction, 1-norm, scalar multiplication are perfectly parallelisable
- Require parallel/distributed matrix–vector multiplication:
  - Graph partitioning
  - Hypergraph partitioning
- Parallel graph partitioners exist
- No existing open-source parallel hypergraph partitioners

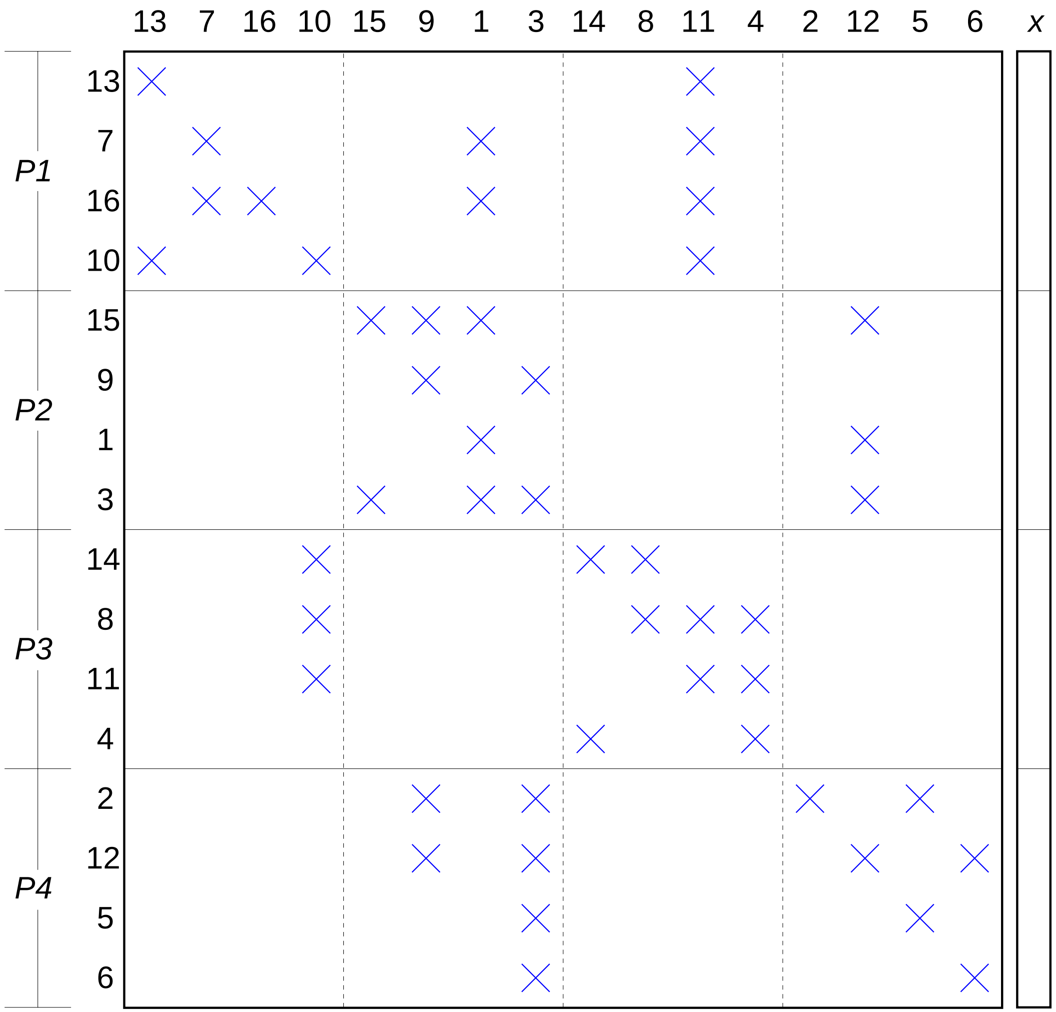
Hypergraph Research

- Currently done at DoC:
  - Will Knottenbelt
  - Nick Dingle
  - Alex Trifunovic
- Graph partitioning balances computational load
- Hypergraph partitioning minimises communication overhead as well

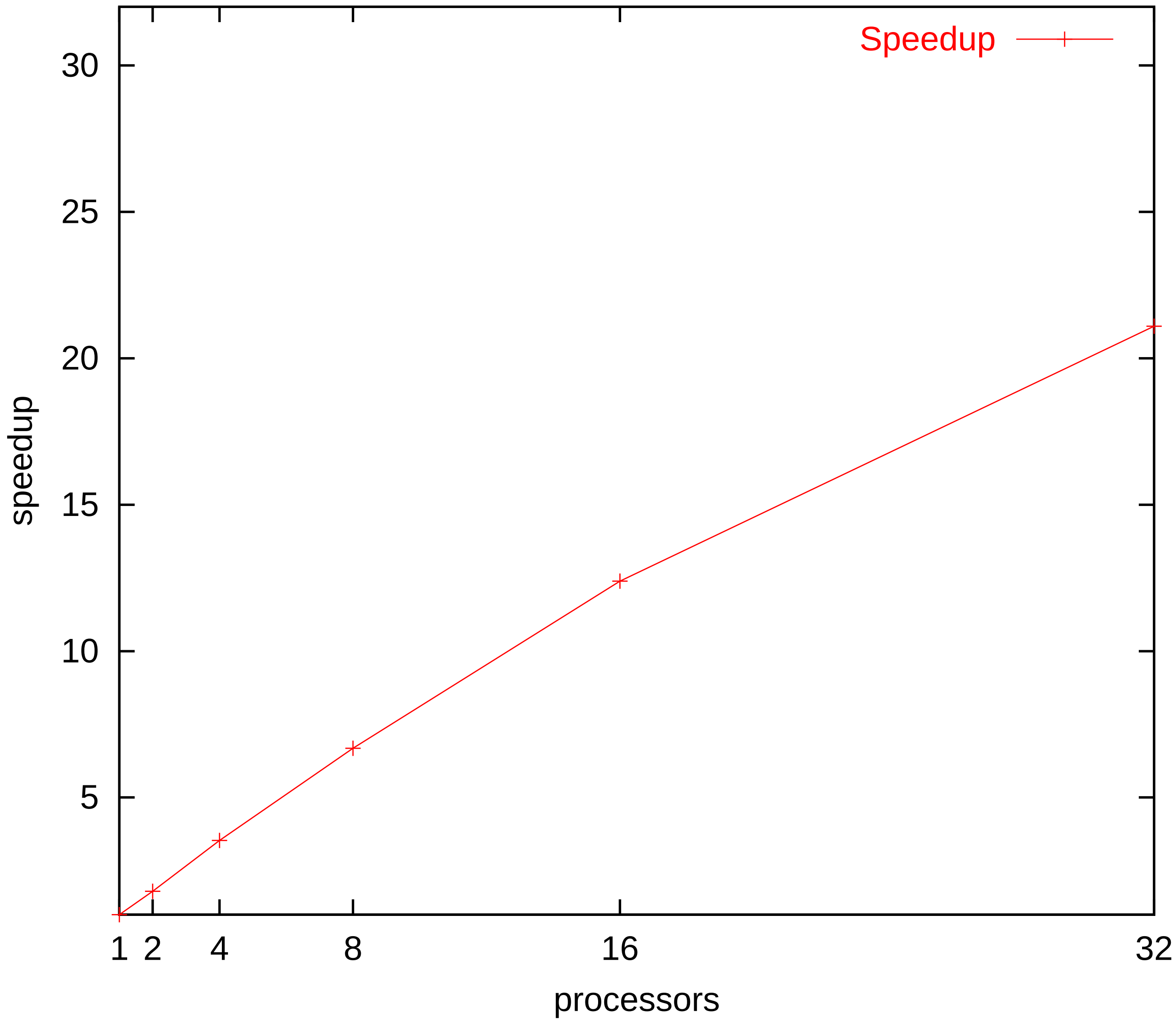
Unpartitioned Graph

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	X											X				
2		X	X		X				X							
3	X		X									X			X	
4				X										X		
5			X		X											
6			X			X										
7	X						X				X					
8				X				X		X	X					
9			X						X							
10										X	X		X			
11				X						X	X					
12			X			X			X			X				
13											X		X			
14								X		X				X		
15	X								X			X			X	
16	X						X				X					X

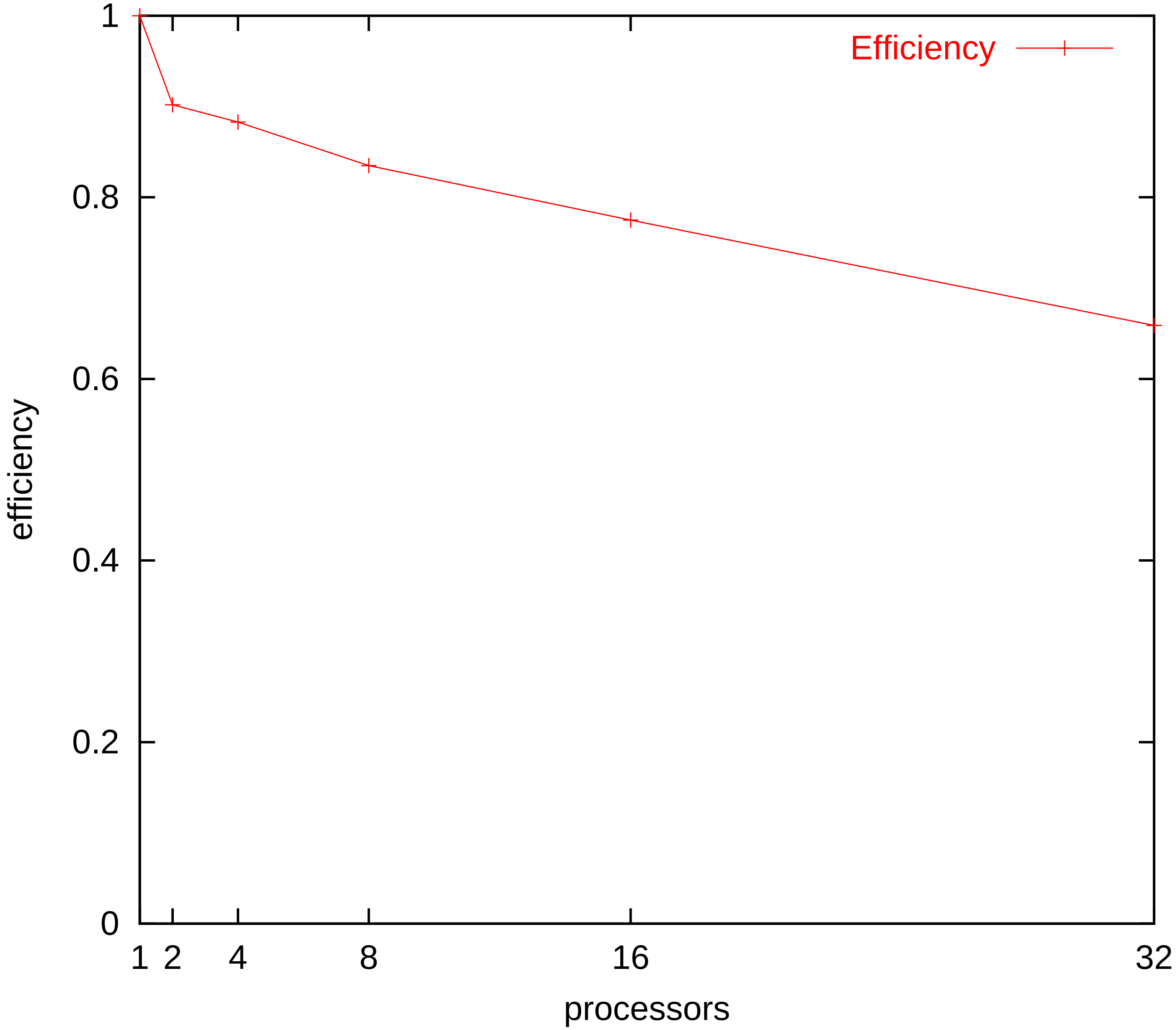
Hypergraph Partition



Speedup over 32 Processors



Efficiency over 32 Processors



Where Next?

- Web as a Peer-to-peer network, a distributed database of documents
- Web servers keep track of own PageRank statistics

$\Rightarrow$ Distributed development of PageRank algorithm (see proposed student project)

  - `http://www.doc.ic.ac.uk/~jb/projects.html`

- BUT... harder to guarantee:
  - availability
  - response-time of query

Acknowledgements

- For comments, discussions and keeping the course on the architectural straight and narrow – thanks go to:
    - Paul Kelly
    - Nick Dingle
    - Ashok Argent-Katwala
    - Tony Field
    - Olav Beckmann
    - Jeyarajan Thiyagalingam
    - all the students who took 332 and who asked insightful questions!