#DOC429 - Study Notes for Parallel Algorithms

Paul Gribelyuk (pg1312, a5)

May 15, 2013

1 Metrics (64 pages)

1.1 Architectures

- *Message passing* used in the case of many machines having only local memory.
- *Shared address space* used when multiple processors in same computer access same memory; *UMA* has equal access times for all, otherwise *NUMA*, so address space is distributed among processors
- *Interconnect Network (IN)* provides hardware to pass messages; topology defines performance: ring, mesh, hypercube

PRAM is an idealization of shared memory MIMD with UMA. Different access modes:

- EREW - exclusive read exclusive write; minimizes concurrency
- CREW - concurrent read exclusive write
- CRCW - concurrent read, concurrent write; to write concurrently, semantics can be *common*, *arbitrary*, *priority*, *reduce* (sum or max or some other reduce operation).
- ERCW - dumb

1.2 Embedding

Binary grey codes used to convert ring network to hypercube:

$$G(0,1) = 0 \tag{1}$$

$$G(1,1) = 1 \tag{2}$$

$$G(i,n+1) = \begin{cases} G(i,n) & i < 2^n \\ 2^n + G(2^{n+1}-1-i,n) & i \geq 2^n \end{cases} \tag{3}$$

Mesh to hypercube can be done by concatenating RGC of each dimension. For node i in $2^{r_1} \times \ldots \times 2^{r_m}$ mesh, mapping is $G(i_1,r_1)G(i_2,r_2)\ldots G(i_m,r_m)$. Can also map a tree to a hypercube. At each level k of the tree, assign the k-th bit either 0 or 1.

1.3 Communication patterns

- Simple message
- One-to-All broadcast; dual is single node accumulation
- All-to-All broadcast; dual is multi-node accumulation
- One-to-One personalized; dual is single node gather
- All-to-All personalized scatter, dual is multi-node gather
- Other (?)

1.4 Performance

- Run Time: T_p
- Speedup: $S_p = \frac{\text{best serial } T}{T_p}$
- Efficiency: $E_p = \frac{S_p}{p}$ is speedup per processor, usually less than 1
- Cost: $C_p = p \cdot T_p$, total amount of computation done on p processors. Cost optimal if $E_p = \Theta(1)$
-

Cost optimality means costs are quivalent to best serial runtime!
Example: to add n numbers on hypercube with $p = 2^d$ nodes each nodes doing $k = n/p$ serial steps, then partial sums reduced via single node accumulation:

$$\begin{aligned} \text{best serial } T &= n && (4)\\ T_p &= \frac{n}{p} + \log p && (5)\\ S_p &= \frac{p}{\frac{n}{p} + \log(p)} && (6)\\ E_p &= \Theta(1/(n/p + \log(p))) && (7)\\ C_p &= n + p\log(p) && (8) \end{aligned}$$

So cost-optimal if $n = \Theta(p\log(p))$.
To understand how algorithm scales, we look at *isoefficiency*. $O_p = C_p - Work$ is a measure of communication latency.

1.5 Communication Costs

- *startup time* t_s incurred once per message
- *per-hop time* t_h
- *transfer time* t_w

Store and forward messages:

$$t_{comm} = t_s + (mt_w + t_h)l$$

Cut-through routing:

$$t_{comm} = t_s + mt_w + lt_h$$

- *Diameter* is maximum length between any two nodes
- *Arc connectivity* is how many links must be broken to fragment network
- *Bisection width* is how many links must be broken to split network into 2 equal halves
- *cost* is usually the number of links in a network

topology	diameter	bisection	arc con	cost
completely connected	1	$p^2/4$	p-1	$p(p-1)/2$
star	2	1	1	p-1
binary tree	$2\log(p+1)/2$	1	1	p-1
linear array	$p-1$	1	1	$p-1$
2-D mesh w/o wrap	$2(\sqrt{p}-1)$	$\sqrt{p}$	2	$2p\sqrt{p-1}$
2-D wraparound	$2\lfloor\sqrt{p}/2\rfloor$	$2\sqrt{p}$	4	$2p$
hypercube	$logp$	$p/2$	$logp$	$p\log p/2$
wrap k-ary d cube	$d\lfloor k/2\rfloor$	$2k^{d-1}$	2d	dp

Different costs associated with these topologies:

operation	hypercube	mesh	ring
one-to-all bcast	$\min\{(t_s+t_wm)\log(p), 2(t_s\log(p)+t_wm)\}$	$2(t_s+t_wm)\log(p)$	$t_s+t_wm\log(p)$
all-to-all bcast	$t_s\log(p)+t_wm(p-1)$	$(t_s+t_wm\sqrt{p})(\sqrt{p}-1)$	$(t_s+t_wm)(p-1)$
all-reduce	$\min\{(t_s+t_wm)\log(p), 2(t_s\log p+t_wm)\}$		
scatter, gather	$t_slogp+t_w(p-1)$		
ATA personalized	$(t_s+t_wm)(p-1)$		
circular shift	t_s+t_wm		

Isoefficiency is how E scales with amount of work W and with p. Goal is to find a way to scale W as a function of p.

$$T_p = \frac{W+O(W,p)}{p} \quad \implies S_p = \frac{W}{T_p} = \frac{W\cdot p}{W+O(W,p)} \quad \implies E = \frac{S}{p} = \frac{1}{1+O(W,p)/W}$$

Another way to formulate cost-optimality:

$$pT_p = \Theta(W) \quad \implies W+O(W,p) = \Theta(W) \quad \implies W = \Omega(O(W,p))$$

2 Dense Matrix (49 pages)

2.1 Matrix-Vector Multiply

Serial runtime for this is $\Theta(n^2)$. Let each processor have a row and an element in the vector. Need all-to-all broadcast of the vector element from each processor to all other processors:

$$T_{comm-hypercube} = t_s\log p + t_w(p-1)$$

$$T_{comm-mesh} = t_s\sqrt{p} + t_w p$$
$$T_{proc} = n$$
$$T_{total} = n + T^1_{comm} = \Theta(n)$$

This is cost-optimal because $pT_p = \Theta(n^2) = T_{serial}$ Assign $k = n/p$ rows to each processor:

$$T_{comm-hypercube} = t_s \log p + t_w \frac{n}{p}(p-1) = t_s \log p + t_w n$$
$$T_{comm-mesh} = t_s\sqrt{p} + t_w \frac{n}{p} p = t_s\sqrt{p} + t_w n$$
$$T_{proc} = \frac{n^2}{p}$$
$$T_{total} = T_{proc} + T_{comm} = \Theta(n^2)$$

To ascertain scalability, compute overhead:

$$O_p(mesh) = C_p - W = pT_{total} - W = t_s p^{\frac{3}{2}} + t_w np$$

For the t_s term only, $W = Kt_s p\sqrt{p}$. For the t_w term only, $W = Kt_w np$, so:

$$W = n^2 = Kt_w np \quad \Longrightarrow n = Kt_w p \quad \Longrightarrow n^2 = K^2 t_w^2 p^2 = W$$

For the hypercube, we get:

$$O_p(hypercube) = C_p - W = pT_{total} - W = t_s p \log p + t_w np$$

So $W = Kt_s p \log(p)$ or $W = K^2 t_w^2 p^2$. We can also do 2D partitioning for matrix-vector multiply. Try $p = n^2$, then need to broadcast vector to diagonals ($p_{i,i}$), broadcast vector element from diagonal to rest of elements in column, then a single multiply operation, then a an all-to-one reduction with sum along the rows:

$$T^1_{comm-hypercube} = T^2_{comm-hypercube} = t_s + t_w \log n \quad (9)$$
$$T^3_{comm-hypercube} = \min\{(t_s + t_w)\log n, 2(t_s \log n + t_w)\} \quad (10)$$
$$T_{total} = 3 * \Theta(\log n) + \Theta(1) \quad (11)$$
$$pT_p = \Theta(n^2 \log n) > \Theta(n^2) \quad (12)$$

This is not cost-optimal. However, we can split up the matrix into blocks of $n/\sqrt{p} \times n/\sqrt{p}$. Still need to send vector elements to diagonal blocks, broadcast from diagonal blocks to all blocks in same column, do a local computation $\Theta(n^2/p)$ and then all-to-one reduce in each row of blocks.

$$T^1_{comm-mesh} = t_s + t_w n/\sqrt{p} \quad (13)$$
$$T^2_{comm-mesh} = T^4_{comm-mesh} = (t_s + t_w \frac{n}{\sqrt{p}})\log\sqrt{p} \quad (14)$$
$$T^3_{proc} = \frac{n^2}{p} \quad (15)$$

For hypercube topologywe get the same thing and scalability is:

$$O_p = pT_p - W = t_s p \log\sqrt{p} + t_w n\sqrt{p}\log\sqrt{p}$$

The isoefficiency terms are: $W = KT_s p \log\sqrt{p}$ and $W = K^2 t_w^2 p \log^2\sqrt{p}$. Therefore, W can grow as $p\log^2 p$ therefore $\log p + 2\log\log p$ can grow as $\log n$. Keeping higher order terms and substituting back, we get that $p\log^2 n = O(n^2)$ so $p = O(\frac{n^2}{\log^2 n})$ is the number of processes which can be used cost-optimally for a given n.

2.2 Matrix Transpose

If we have $p = n^2$, then need to move each element at most $2(n-1)$ steps, which takes $2(n-1)(t_s + t_h + t_w)$. If we have $p < n$ then assign blocks of $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$ of the matrix to each processor. Each one does the local transpose serially, yielding time of $\frac{n^2}{2p}$, then the p blocks need to be moved around into the right place, which takes:

$$T_{comm} = 2(\sqrt{p}-1)(t_s + t_w n^2/p) \quad \Longrightarrow C_p = pT_p = \frac{n^2}{2} + 2t_s p\sqrt{p} + 2t_w n^2\sqrt{p}$$

Not cost optimal.

2.3 Matrix-Matrix Multiply

Best serial runtime:

$$T_{serial} = \Theta(n^3)$$

Each process can own a $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$ component of **A** and **B**, writing results into **C**. Each process will send its $\frac{n^2}{p}$ **A**-values to all other blocks in its row of **A** and all other **B**-values down the rows of **B**. Each process does a local submatrix multiply, costing $\frac{n}{\sqrt{p}} \cdot \frac{n}{\sqrt{p}} \cdot n = \frac{n^3}{p}$. Communication costs are:

$$T_{comm-mesh} = 2\left(t_s\sqrt{p} + t_w \frac{n^2}{p}(\sqrt{p}-1)\right)$$

So total runtime is $n^3/p + 2t_s\sqrt{p} + 2t_w\frac{n^2}{p}$ so cost-optimal if $p = O(n^2)$. To get isoefficiency, substitute $W^{\frac{2}{3}} = n^2$ into the cost equation and solve:

$$W = 2Kt_s p^{\frac{3}{2}} \tag{16}$$

$$\text{or} \quad W = 8K^3 p^{\frac{3}{2}} t_w^3 \tag{17}$$

Cannon's algorithm tries to save on space by performing shifts of the submatrices after each computation. *DNS algorithm* uses n^3 processes, broadcasting the bottom matrix rowwise to the level that is the index of that row for both **A** and **B**. It then multiplies the values, and sums them down the vertial axis.

$$T_p = \frac{n^3}{q^3} + 3t_s \log q + 3t_w \frac{n^2}{q^2} \log q$$

where $q = p^{1/3}$. Then cost-optimal for $p = O(n^3/(\log n)^3)$ and isoefficiency is $\Theta(p(\log p)^3)$

3 Linear Equations (21 pages)

Trying to solve $\mathbf{Ax} = \mathbf{b}$. Gaussian elmination costs $\Theta(n^3)$. Let's represent **A**:

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$$

as sum of lower, diagonal, and upper matrices. Jacobi equation:

$$x = \mathbf{D}^{-1}(\mathbf{b} - (\mathbf{L} + \mathbf{U})\mathbf{x})$$

so can update iteratively:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{i \neq j} a_{ij} x_j^{(k)})$$

Gauss-Seidel method says that since we know $x_i^{(k+1)}$ by the time we're applying it to **L**, we can write instead:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)})$$

Gaussian elimination is not cost-optimal in the usual implementation, but can be made so by pipelining processors, so they do work when it's ready for them to do if they don't need to wait. Time between iterations is:

$$t_s + (t_w + 3t_a)(n - k - 1)$$

There is a still a load imbalance which can be solved by block-cyclic striping.

4 Partitioning (27 pages)

4.1 1D Graphs

Each row i is a vertex v_i in the graph. Weight w_i is number of non-zero elements in row i. Edge weight on e_{ij} connecting v_i to v_j is 1 if $a_{ij} \neq 0$ OR $a_{ji} \neq 0$ and 2 if they are both zero. Trying to pick partition which minimizes *edge cut*. An edge e_{ij} is cut if v_j belongs to different partition than v_i. Let:

$$W_k = \sum_{i \in P_k} w_i$$

is sum of weights of partition P_k. if $\bar{W}$ is average weight, then partition balanced if:

$$|W_k - \bar{W}| < (1 + \epsilon) \quad \forall k$$

Computational load is all non-zero elements in the partition. Communication volume is how many vector elements will need to be sent to each processor. 1D graph does not minimize this well.

4.2 1D Hypergraphs

A hypergraph holds vertices and *nets*. Each row i is a vertex v_i and each column j is a net n_j. The net n_j holds all vertices v_i such that $a_{ij} \neq 0$. The *connecivity* λ_j of net n_j is number of different partitions to which elements of n_j belong. The cutsize is $\sum_{n_j}(\lambda_j - 1)$. This does a better job at minimizing communications because it knows that two non-zero elements in the same column will not need two sends of the corresponding vector element.

4.3 2D Hypergraph

Poorly explained/understood...

5 Search (73 pages)

- DFS
- DFBB (discard paths which are worst than best current solution)

- ID-DFS (limit on depth)
- IDA* (discard paths lower than dynamically specified lower bound)

Need work-splitting strategies.

- Asynchronous Round Robin
- Global Round Robin
- Random Polling

6 MPI (31 pages)

```
MPI_Init()
MPI_Finalize()
MPI_Comm_rank()
MPI_Comm_size()
MPI_Send()     (blocking)
MPI_Recv()     (blocking)
MPI_Isend()    (non-blocking)
MPI_Irecv()    (non-blocking)
MPI_Wait()     (wait for send/receive completion)
MPI_Waitall()
MPI_Iprobe()   (see if message is there)
MPI_Bcast()
MPI_Scatter()  (one to all personalized)
MPI_Gather()   (all to one personalized)
MPI_Reduce()
MPI_Allgather(), MPI_Allreduce()
MPI_Barrier()  (synchronize processes)
MPI_Wtime()
```