

# CS 442 Introduction to Parallel Processing

## Homework 1

Erik Erhardt

February 10, 2006

**GGKK 2.1** *Discuss how applicable the STREAMS benchmark would be to answering question 2.1*

STREAM is a test of memory bandwidth. Each of four operations work on arrays which are larger than the cache, and the code does not allow data reuse. Thus, for a fixed number of FLOPs, one obtains a measure of the memory bandwidth.

Dealing with Memory Hierarchies is a future direction for STREAM, and is under evaluation as STREAM2.

**GGKK 2.2** *dot product*

Given 100 cycles for fetch of 4 words, lay out vectors linearly in memory. 8 FLOPs (4  $\times$ , +) in 200 cycles leads to  $\frac{8}{200} = 40$  MFLOPS. Note, it takes  $\lceil \frac{dim}{4} \rceil \times 200$  cycles for computation.

**GGKK 2.3** *matrix-vector multiplication*

Read in the vector and the first row of the matrix, and keep vector in cache for reuse.  $200 \times K$  cycles for first row of matrix, 8 FLOPs (4  $\times$ , +) in each 200 cycles. For remaining  $K - 1$  rows,  $100K(K - 1)$  cycles, 8 FLOPs in each 100 cycles. It takes a total of  $K(200 + 100(K - 1)) = 100K^2 + 100K$  cycles, with  $8K^2$  FLOPs.  $\frac{8K^2}{100K^2 + 100K} = \frac{8}{100 + 100/K}$  FLOPS goes to 80 MFLOPS as  $K$  increases to half of cache size for vector.

**GGKK 2.14** *d-dimensional hypercube, Hamming distance and parallel paths*

Let  $H(A, B)$  be the Hamming distance between  $A$  and  $B$  and let  $P(A, B)$  be the number of distinct paths between  $A$  and  $B$ .

Represent the nodes in binary in the customary way.

1. *minimum distance.* Show  $\min(\text{dist}(A, B)) = H(A, B)$ .

Consider transitioning between connecting nodes,  $N_1$  to  $N_2$ . That transition can be represented by switching the binary switch indicating movement from the parallel  $d - 1$  dimension hypercube  $N_1$  is in to that of  $N_2$ . Not only are these transitions the minimum number, since it makes no sense to switch a switch more than once, the bits that differ is the set of unordered transitions necessary to move from  $A$  to  $B$ .

2. *total parallel paths.* Show  $P(A, B) = d$ .

The degree of each node is  $d$ , thus there are no more than  $d$  parallel paths.

Constructed ineffectively, it is easily possible to achieve less than  $d$  parallel paths. To achieve  $P(A, B) = d$ , first take the  $H(A, B)$  shortest paths of length  $H(A, B)$ , then the paths of length  $H(A, B) + 2$ .

3. *number of parallel paths of length  $H(A, B)$ .* Show  $P_{\text{length}=H(A,B)}(A, B) = H(A, B)$ .

The subset of bits identified by  $H(A, B)$  define a slice of the hypercube, with dimension  $H(A, B) = d_H$ . By part (2), there are  $d_H$  parallel paths in this sub-hypercube.

4. *remaining parallel paths.* Show  $P_{\text{length}=H(A,B)}(A, B) = H(A, B)$ .

For the remaining  $d - H(A, B)$  parallel paths, it is required to make exactly one transition from the sub-hypercube in (3) and exactly one transition back to have the shortest length — this is equivalent to switching a matching bit between  $A$  and  $B$  twice. The first transition moves between two parallel hypercubes of dimension  $d - 1$ , then makes equivalent moves as shortest paths in  $H(A, B)$ , then returns.

Note that it is possible to have longer paths. For example, consider a  $d = 3$  cube, moving between adjacent nodes. There is one path of length 1, then one path of length 3, then the third path has the option of length 3 or length 5.

**GGKK 2.23** *minimum-dilation from binary tree to hypercube*

Minimum dilation is 2 for a  $2^d - 1$  binary tree onto  $d$ -dimension hypercube, where the dilation occurs at the root. The number of edges for the root links to the second-level nodes is 1 for one link and 2 for the second link. The lower nodes and edges beyond the root map readily onto the hypercube.

**GGKK 2.26** *store-and-forward*


---

$t_s$ =startup time
$t_w$ =per-word transfer time
$d$ =distance in number of hops
$m$ =message size
$k$ =number of message parts

---

**1.** *send when previous reaches next node*

There will be  $k$  start up times, and each packet will take  $t_w \frac{m}{k} d$  time to reach destination, with a trail of  $k$  of them (so  $(d + k - 1)$ ).

$$\text{time} = kt_s + t_w \frac{m}{k} (d + k - 1)$$

**2.** *send when previous reaches destination*

There will be  $k$  start up times, and each packet will take  $t_w \frac{m}{k} d$  time to reach destination, but they do not share travel time (so  $(dk)$ ).

$$\text{time} = kt_s + t_w \frac{m}{k} (dk)$$

$k$	method 1	method 1
1	$1t_s + t_w \frac{m}{1} (d + 1 - 1) = t_s + t_w md$	$1t_s + t_w \frac{m}{1} (d1) = t_s + t_w md$
...		
$m$	$mt_s + t_w \frac{m}{m} (d + m - 1) = mt_s + t_w (d + m - 1)$	$mt_s + t_w \frac{m}{m} (dm) = mt_s + t_w (dm)$
$t_s$ large	set $k = 1$	set $k = 1$
$t_s = 0$	set $k = m$	$k$ inconsequential

When  $k = 1$ , these methods are equivalent. When  $k = m$  they are very different, especially as the message length grows.