Energy Savings through Embedded Processing on Disk System

S. W. Son, G. Chen, M. Kandemir, F. Li*

Dept of Computer Science & Engineering

The Pennsylvania State University

ASPDAC'06

Outline

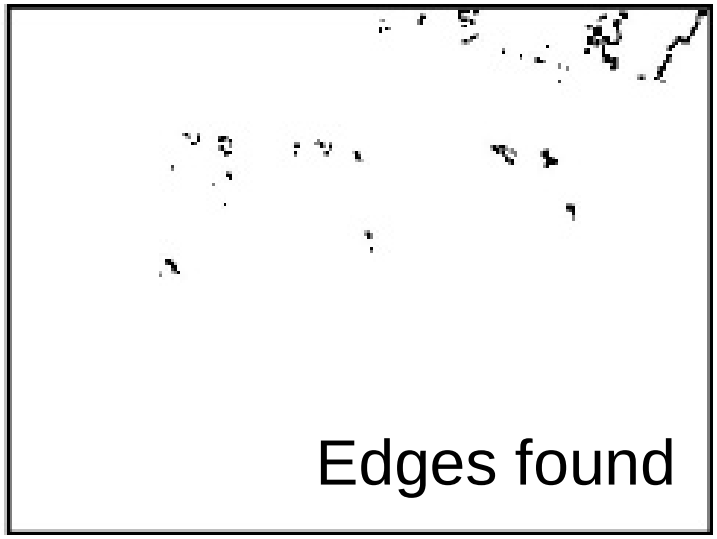
- Motivation
- Smart Disk Architecture
- Related Work
- Our Approach
- Experimental Setup and Results
- Conclusion

Motivation

- Many data-intensive applications are tightly coupled with disk subsystem
  - Computations that depend on disk data are filtering type
- Smart disks: embedding computing power in the storage devices
  - Performing computing in the storage device instead of transforming large data sets to the host
  - Addressing the huge I/O demands for the next generation applications

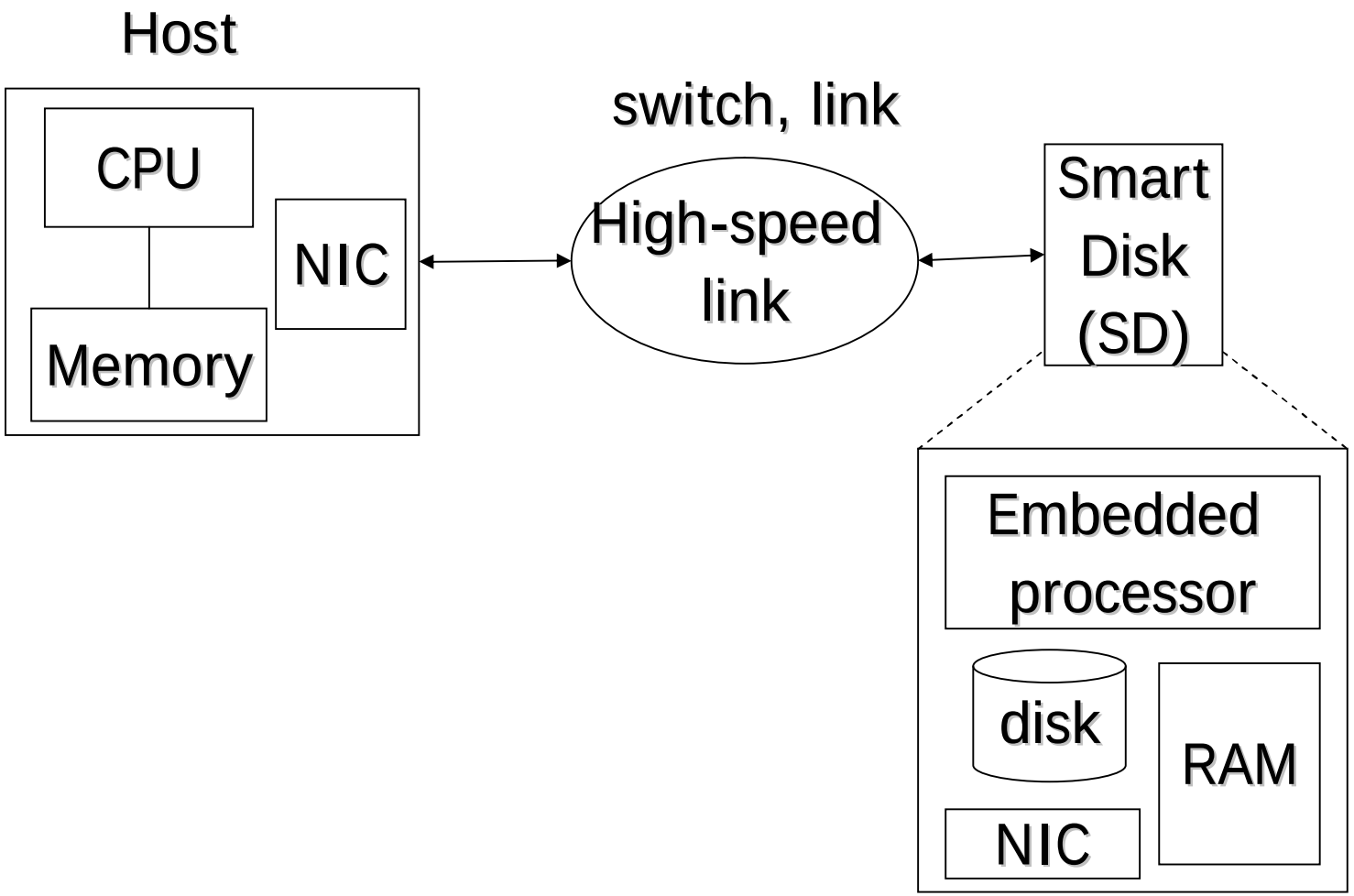
What is Filtering Type?

- Using the SD system, edge detection for each image is performed directly at the drives and only the edges are returned to the HOST.
- A request for the raw image at the left returns only the data on the right, which is much more compact.



Source: IBM Almaden's CattleCam

Smart Disk Architecture

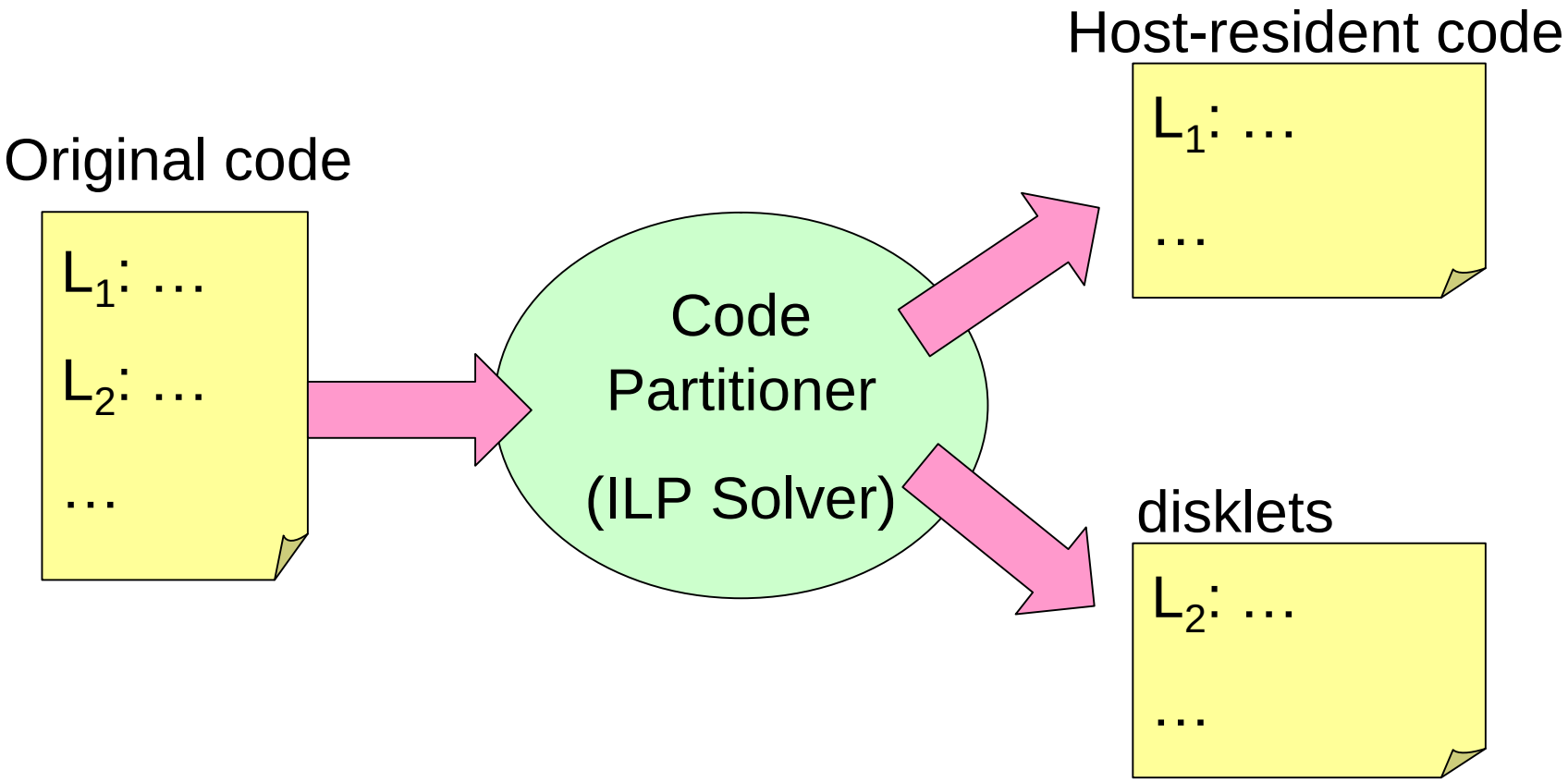


Related Work

- Embedded processing on the disk/memory subsystem
  - Active/smart disk : [Acharya et al], [Riedel et al], [Uysal et al], [Chiu et al] and [Memik et al]
  - ISTORAGE : UC Berkeley
  - IRAM and PIM: UC Berkeley and Univ. of Notre Dame
- Compiler-based code partitioning for enhancing performance [G. Chen et al]

→ This paper focuses on the code partitioning for energy savings through embedded processing

Our Approach



Our Approach

- Compiler divides a given code fragment into two parts:
  - Host-resident codes
  - Disklets
- We use ILP formulation to determine the optimal execution strategy for the given program
  - Goal is to minimize the total energy consumed by the program

ILP Formulation

- Variables determined by the compiler
  - $J_{i,j}$: $J_{i,j}$ = 1, if arrays A_i and A_j share some elements
  - N_i : number of iterations for loop nest L_i
  - X_i, E_i : time/energy per iteration for executing L_i on the HOST
  - X_i', E_i' : time/energy per iteration for executing L_i on the SD
  - $W_{i,j}$: 1 if L_j updates the array elements of A_i
  - $R_{i,j}$: 1 if L_j reads the array elements of A_i
- Variables determined by ILP solver
  - H_i : 1 if L_i is assigned to HOST, otherwise 0
  - $M_{i,j}$: 1 if A_i is in the HOST memory initially
  - $D_{i,j}$: 1 if A_i is dirty at the entry of L_j

ILP Formulation – cont'd

$$E_{leakage} = P\sum_{j=1}^{n}(H_j T_j + (1-H_j)T_j')$$

$$E_{dynamic} = \sum_{j=1}^{n}(H_j N_j E_j + (1-H_j)N_j E_j')$$

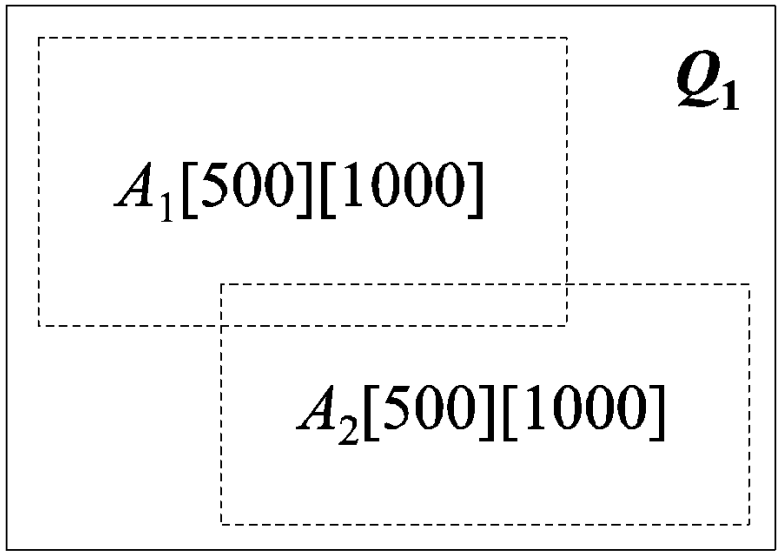
$$E_{link} = \sum_{j=1}^{n} H_j E_j^*$$

$$E = E_{link} + E_{leakage} + E_{dynamic}$$

Example

```
L1: for i = 0 to 999
        for j = 0 to 499
            A1[i][j] = g (A3[i],j);
L2: for i = 0 to 999
        for j = 0 to 499
            A3[i] = A3[i] + A2[i][j];
L3: for i = 0 to 999
        A3[i] = h (A3[i]);
```

(a) Original code



(b) Array layouts

Example – cont'd

$J_{i,j}$ (i →, j ↓)

j \ i	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

$W_{i,j}$ (j →, i ↓)

i \ j	1	2	3
1	1	0	0
2	0	0	0
3	0	1	1

$R_{i,j}$ (j →, i ↓)

i \ j	1	2	3
1	1	0	0
2	0	1	0
3	1	1	1

i	N_i	X_i	X_i'
1	500,000	100	800
2	500,000	10	80
3	1000	200	1600

* These variables are determined statically by the compiler

Example – results of the ILP solver

i	H_i
1	1
2	0
3	1

$M_{i,j}$

i \ j	1	2	3
1	0	0	0
2	0	0	0
3	0	1	1

$D_{i,j}$

i \ j	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

Example – cont'd

L_1: for i = 0 to 999

for j = 0 to 499

$A_1[i][j] = g(A_3[i],j)$;

write A1 back to disk;

signal SD to start L_2;

wait for signal;

load A_3 into memory;

L_3: for i = 0 to 999

$A_3[i] = h(A_3[i])$;

wait for signal;

L_2: for i = 0 to 999

for j = 0 to 499

$A_3[i] = A_3[i] + A_2[i][j]$;

signal end of L_2;

Default Simulation Parameters

- HOST Processor: Intel P4 2.0GHz
- Embedded Processor: StrongARM 200MHz
- Memory: 32MB for SD and 1GB for HOST
- Disk: IBM Ultrastar 36Z15 (15K RPM)
- Interconnects: Infiniband 1x
- Switch Fabrics: IBM Infiniband 1x switch

- See the paper for details of performance & power values

Benchmarks

Name	Total Data (MB)	Base Energy (J)	Execution Time (sec)	Link Energy (%)	% of code on SD
swim	22.1	736.6	4.4	23.9%	59%
apsi	2.9	101.6	0.6	23.8%	74%
mgrid	80.7	2707.1	16.2	23.6%	54%
bmcm	10.3	457.5	2.6	22.3%	28.3%

* Benchmarks are selected from SPEC2000 and Perfect club

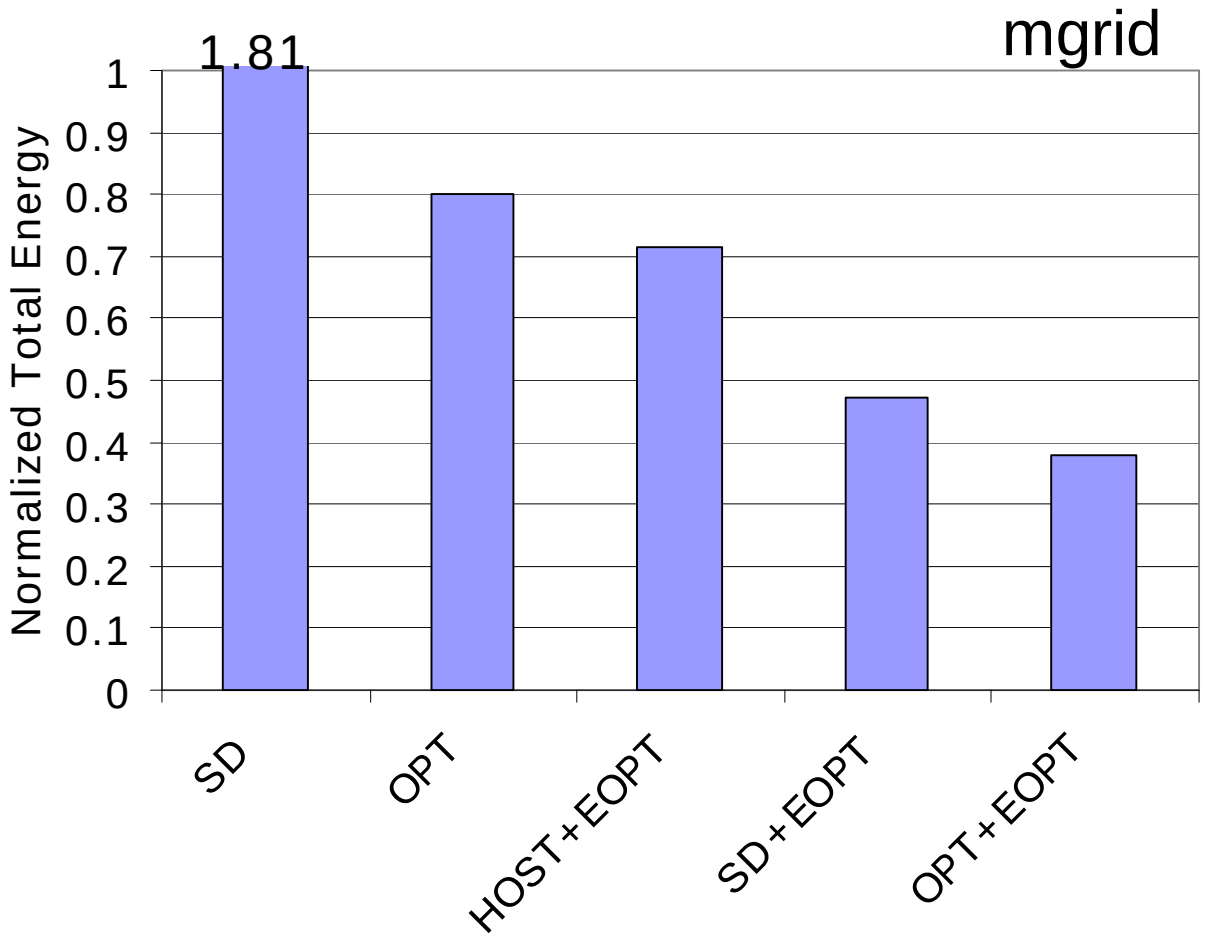
Evaluated Schemes

- HOST: all computations are performed on the host system
- SD: all computations are performed on the smart disk system
- OPT: computations are partitioned based on our approach
- HOST+EOPT: HOST scheme with power control
- SD+EOPT: SD scheme with power control
- OPT+EOPT: OPT scheme with power control

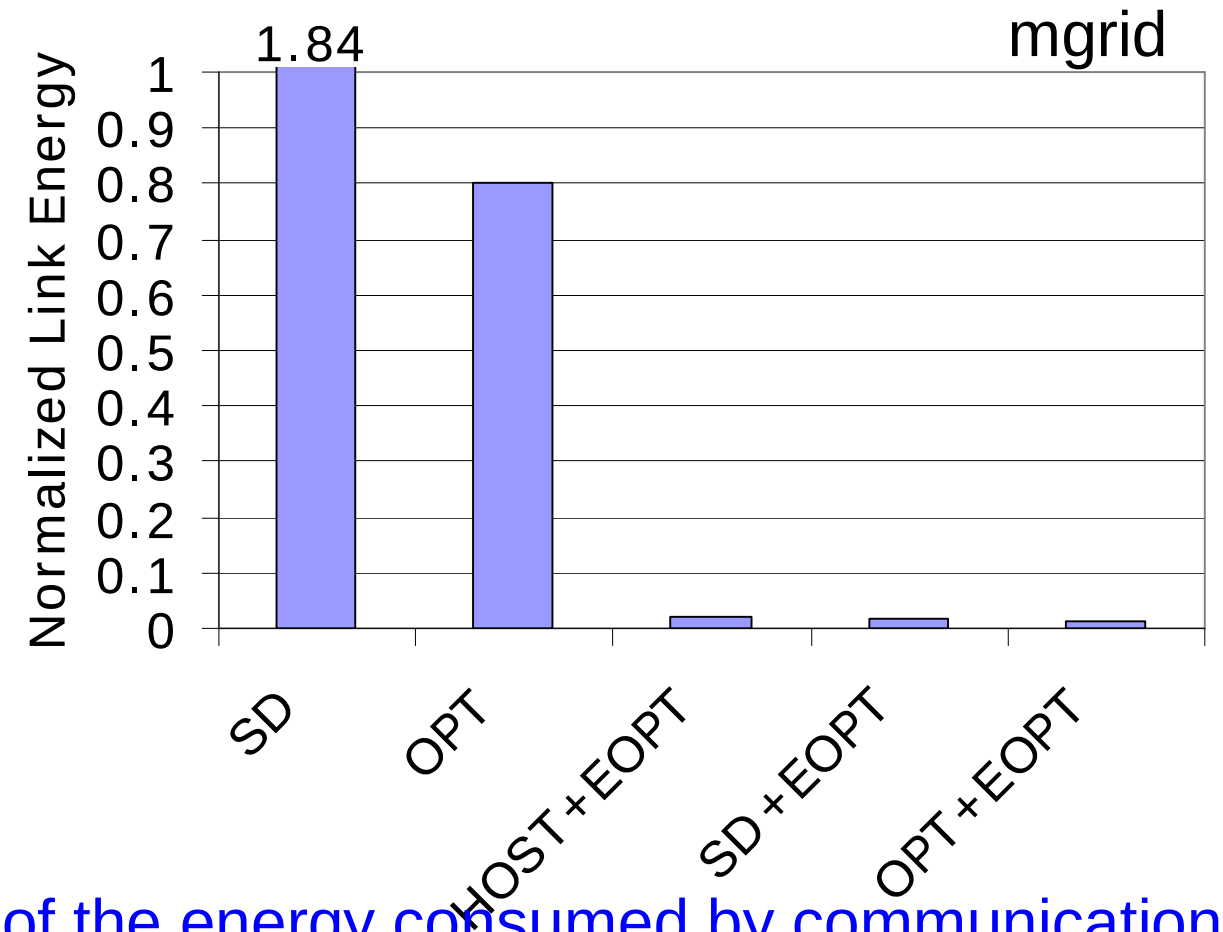
EOPT Scheme

- Each system component can be in a low-power mode when it is not in use
  - e.g., CPU, memory, interconnect, etc
- The decision to place a component in the low-power mode is based on breakeven time of each component

Normalized Total Energy Consumption



Normalized Link Energy Consumption



Most of the energy consumed by communication links can be eliminated if we exploit low power mode

Conclusion

- We propose ILP-based approach that partitions an application code between the host system and the disk system (equipped with an embedded processor and associated memory)
- We experimentally evaluated our approach using a set of array-intensive benchmarks that frequently exercise the disk-resident datasets
- Our experimental results indicate that the proposed partitioning approach reduces power consumption significantly

Thank You!

sson@cse.psu.edu