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

Host

<table>
<thead>
<tr>
<th>CPU</th>
<th>NIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td></td>
</tr>
</tbody>
</table>

switch, link

High-speed link

Smart Disk (SD)

Embedded processor

<table>
<thead>
<tr>
<th>disk</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NIC</td>
<td></td>
</tr>
</tbody>
</table>

Host diagram with components labeled.
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

Original code

\[ L_1: \ldots \]
\[ L_2: \ldots \]
\[ \ldots \]

Code Partitioner (ILP Solver)

Host-resident code

\[ L_1: \ldots \]
\[ \ldots \]

disklets

\[ L_2: \ldots \]
\[ \ldots \]
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

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

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

\[ E_{\text{link}} = \sum_{j=1}^{n} H_j E_j^* \]

\[ E = E_{\text{link}} + E_{\text{leakage}} + E_{\text{dynamic}} \]
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

\[
J_{i,j} \begin{array}{ccc}
i & 1 & 2 & 3 \\
j & 1 & 1 & 1 & 0 \\
 & 2 & 1 & 1 & 0 \\
 & 3 & 0 & 0 & 1 \\
\end{array}
\]

\[
W_{i,j} \begin{array}{ccc}
i & 1 & 0 & 0 \\
j & 0 & 0 & 0 \\
 & 0 & 1 & 1 \\
\end{array}
\]

\[
R_{i,j} \begin{array}{ccc}
i & 1 & 0 & 0 \\
j & 0 & 1 & 0 \\
 & 1 & 1 & 1 \\
\end{array}
\]

<table>
<thead>
<tr>
<th>i</th>
<th>(N_i)</th>
<th>(X_i)</th>
<th>(X'_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>500,000</td>
<td>100</td>
<td>800</td>
</tr>
<tr>
<td>2</td>
<td>500,000</td>
<td>10</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>1000</td>
<td>200</td>
<td>1600</td>
</tr>
</tbody>
</table>

* These variables are determined statically by the compiler
Example – results of the ILP solver

\[
\begin{array}{c|c}
  i & H_i \\
  \hline
  1 & 1 \\
  2 & 0 \\
  3 & 1 \\
\end{array}
\]

\[
 M_{i,j} \begin{array}{ccc}
  j \\
  \hline
  1 & 0 & 0 & 0 \\
  2 & 0 & 0 & 0 \\
  3 & 0 & 1 & 1 \\
\end{array}
\]

\[
 D_{i,j} \begin{array}{ccc}
  j \\
  \hline
  1 & 0 & 0 & 0 \\
  2 & 0 & 0 & 0 \\
  3 & 0 & 0 & 0 \\
\end{array}
\]
Example – cont’d

\[
L_1: \text{for } i = 0 \text{ to } 999 \\
\quad \text{for } j = 0 \text{ to } 499 \\
\quad A_1[i][j] = g(A_3[i], j); \\
\quad \text{write } A1 \text{ back to disk;} \\
\quad \text{signal SD to start } L_2; \\
\quad \text{wait for signal;} \\
\quad \text{load } A_3 \text{ into memory;} \\
L_3: \text{for } i = 0 \text{ to } 999 \\
\quad A_3[i] = h(A_3[i]); \\
\]

\[
\text{wait for signal;} \\
L_2: \text{for } i = 0 \text{ to } 999 \\
\quad \text{for } j = 0 \text{ to } 499 \\
\quad A_3[i] = A_3[i] + A_2[i][j]; \\
\quad \text{signal end of } L_2; \\
\]

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

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

* 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

SD: 1.81
OPT: 0.8
HOST+EOPT: 0.7
SD+EOPT: 0.5
OPT+EOPT: 0.4

mgrid
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