Compiler-Based Code Partitioning for Intelligent Embedded Disk Processing

Guilin Chen, Guangyu Chen, M. Kandemir, A. Nadgir
The Pennsylvania State University
Introduction

- Efforts towards making the disk storage system more intelligent
  - Active disks
  - Intelligent disks
  - Smart disks

- Idea: use computing power at the disk to perform some filtering type of computation on the storage device itself
Prior Studies

- W. Hsu et al, Projecting the Performance of Decision Support Workloads on Systems with Smart Storage (SmartSTOR), Report No. UCB/CSD-99-99-1057, 1999
- Most of them focus on application programming model and OS support
Our Approach

- It is difficult for programmer to divide the workload between disk and host system
- We propose a compiler-based strategy that automatically divides an application
- Focus on image and video applications
- Identify computations that exhibit a **filtering characteristic**
  - Their output data sets are much smaller than their input data sets
Abstract View of the Architecture

Host System

Processor

Memory

Intelligent Disk

Disk System

Embedded Processor

Controller

Memory

Disk(s)
Programming Model

- Two compiler directives that enclose a code portion which will be executed on the disk system
  - \texttt{begin\{map\} ... end\{map\}}
- The above directives are automatically inserted in the application by the compiler
- Operate on a loop nest granularity
- Assume that the disk-resident arrays are annotated using a special compiler directive
Work Division

- Our compiler analyzes the entire application to extract data access pattern, and then inserts `\texttt{begin\{map\}}` and `\texttt{end\{map\}}` calls in the code to perform work division.

- A code fragment is mapped on to the disk system if it satisfies the following two criteria:
  - Performs input/output (I/O)
  - Exhibits a filtering characteristic
Filtering Characteristics

- \texttt{size\_of(Input\ Data)} \gg \texttt{size\_of(Output\ Data)}
- Large input data set means a lot of network traffic can be saved by performing computation in the disk system.
- Small output data set means not much data to transfer from disk to host system once the computation is finished at the disk side.
for $I = 2..N-1$
  
  for $J = 2..N-1$
    
    $V[i][j] = 0.25 \times (U[i][j-1] + U[I][j+1]$
    
    $+ U[I-1][J] + U[I+1][J])$

for $I = 1..N$
  
  for $J = 1..N$
    
    for $K = 2..N$
      
      $X[I][J] = 0.33 \times (W[I][J][K-1] + W[I][J][K]$
      
      $+ W[I][J][K+1])$
Detecting the Filtering Characteristic: Strategy I

for i1 = L1..U1
  for i2 = L2..U2
    ...
      for is = Ls..Us
        U[f1][f2]...[fn] = ... V[g1][g2]...[gm] ...

- Assume that arrays $U^{(n\text{-dimensional})}$ and $V^{(m\text{-dimensional})}$ are declared as type
  $U[N1][N2]...[Nn], V[M1][M2]...[Mm]$

- The assignment statement in the loop shown above exhibits a filtering characteristic if:
  \[ c \times N1 \times N2 \times ... \times Nn < M1 \times M2 \times ... \times Mm \]

- Selecting a suitable value $c$ is critical
Detecting the Filtering Characteristic: Strategy II

- Decides the assignment statement exhibits a filtering characteristic if:
  \[ c \times G\{U[f1][f2]...[fn]\} < G\{V[g1][g2]...[gm]\} \]
- \( G\{E\} \) gives the number of distinct array elements accessed by array reference \( E \)
- Strategy II checks the actual number of elements accessed
- More accurate, but more costly
Reducing Communication

- Mapping large code fragments to the disk system is preferred to mapping smaller ones.
- Two neighboring nodes, frag1 and frag2, should be mapped as a whole, if frag2 is the only consumer of frag1’s output dataset.
- Our current implementation uses data-flow analysis for this purpose.
Parallel Processing on the Disk System

- Compiler support for multiple embedded processors on the disk system
- Additional constraint to map a code fragment onto the disk system
  - The code fragment should take advantage of the parallel embedded processors on the disk system
Simulation Environment

- Use DiskSim for simulating the disk behavior
- The simulation of the host processor(s) and embedded processor(s) are performed under Simplescalar
- For communication, a simple strategy based on the number of communication messages and the available bandwidth
Simulation Process

- Input Program
- Work Division
- Disk Program
- Host Program
- Simple Scalar
- Comm. Simulator
- Simple Scalar
- DiskSim
- Statistics
## Benchmark Codes

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Brief Description</th>
<th>Total Dataset Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature</td>
<td>Feature Extraction</td>
<td>11.72MB</td>
</tr>
<tr>
<td>ImgComp</td>
<td>Image Compression</td>
<td>8.21MB</td>
</tr>
<tr>
<td>Restore</td>
<td>Image Restoration</td>
<td>35.40MB</td>
</tr>
<tr>
<td>SMT</td>
<td>Video Smoothing</td>
<td>17.06MB</td>
</tr>
<tr>
<td>T-Image</td>
<td>Crowd Management with Imaging</td>
<td>16.05MB</td>
</tr>
<tr>
<td>Vehicle-V</td>
<td>Vehicle Tracking and Classification</td>
<td>23.88MB</td>
</tr>
</tbody>
</table>
Normalized Execution Cycles for Different Versions
Normalized Execution Cycles with Different Number of Embedded Processors

![Graph showing normalized execution cycles for different number of embedded processors. The graph includes lines for Feature, ImgComp, Restore, SMT, T-Image, and Vehicle-V, with cycles decreasing as the number of processors increases.]
Conclusion

- Intelligent disk systems with large storage capacities and fast interconnects are expected to become prevalent in the near future.

- An important problem that needs to be addressed in such architectures is how to divide the computation between the disk system and the host system.

- Our approach based on filtering characteristic is promising.
Thanks!