Compiler-Directed Array Interleaving for Reducing Energy in Multi-Bank Memories

V. De La Luz, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, A. Sivasubramanian, and I. Kolcu

Microsystems Design Lab
Pennsylvania State University

www.cse.psu.edu/~mdl
**Motivation**

- Increasing awareness in energy consumption
- Array-dominated applications exhibit poor memory energy behavior
- Multi-bank architectures might help, but
  - We need to make sure that not many banks are simultaneously active
  - Compiler can help
    - Computation transformations
    - Data transformations
Major Contributions

- A compiler framework to map multiple array variables into a single data space (mechanism)
- A global strategy that decides how arrays should be grouped (policy)
- Experimental results on a multi-bank memory architecture showing energy benefits
Multi-Bank Memory Architecture

Self-Monitoring & Prediction

Memory Controller

To/From CPU
Operating Modes

- Active: 3.57 nJ
- Standby: 0.83 nJ
- Nap: 0.32 nJ
- PowerDown: 0.005 nJ
- Disabled: 0 nJ

[Cycles]
- 2 cycles
- 30 cycles
- 9000 cycles

[HPCA’01]
Rambus DRAM (RDRAM)

- High bandwidth (>1.5 GB/sec)
- Each RDRAM module can be activated/deactivated independently
- Read/write can occur only in Active mode
- Three low-power operating modes:
  - Standby, Nap, PowerDown
Array Interleaving

for $I = 1, N$

$b += U_1[I] + U_2[I];$

for $I = 1, N$

$b += X[2I-1] + X[2I];$

Mappings:

$U_1[I] \rightarrow X[2I-1]$ \& $U_2[I] \rightarrow X[2I]$
Array Interleaving

for $I = 1, N$
for $J = 1, N$
    $b += U_1[I][J] + U_2[I][J];$

for $I = 1, N$
for $J = 1, N$
    $b += X[I][2J-1] + X[I][2J];$

Mappings:

$U_1[I][J] \rightarrow X[I][2J-1]$ & $U_2[I][J] \rightarrow X[I][2J]$
Array Interleaving
Interleaving Policy

- **Objective:** Optimizing memory energy by improving bank locality
- **Representation:** Graph-based, uses the concept of maximum-length disjoint paths (path cover)
- An NP-hard problem; but, effective polynomial-time heuristics exist
- Interleaving mechanism defined in [CC’01]
Example - I

for I = 1, N
    \( U_5[I] = (U_3[I]*U_4[I]) + (U_1[I]+U_2[I]) \); 
for I = 1, N
    \( U_6[I] = (U_1[I]+U_3[I]) * U_5[I] \); 
for I = 1, N
    \( U_4[I] = (U_5[I]-U_6[I]) * U_1[I] \); 

Mapped to an array transition graph:
- Nodes: arrays in the code
- Edges: transitions between arrays (references)
Example - II

Diagram:

- U1
- U2
- U3
- U4
- U5
- U6

Connections:
- U1 to U2: N
- U2 to U3: N
- U3 to U5: 2N-1
- U4 to U1: 2N
- U5 to U6: 2N
- U2 to U5: 2N-1
- U3 to U4: N-1

Mathematical expressions:
- 2N
- N
- 2N-1
Example - III

Diagram:

- U1
- U2
- U3
- U4
- U5
- U6

Connections:
- U1 to U4: 2N
- U4 to U2: 2N
- U2 to U3: N
- U3 to U5: 2N-1
- U5 to U6: 2N
- U6 to U5: N-1
Transformed Code:

```c
for I = 1, N
    X[5I-1] = (X[5I]*X[5I-4]) + (X[5I-3]+U_2[I]);
for I = 1, N
    X[5I-2] = (X[5I-3]+X[5I]) * X[5I-1];
for I = 1, N
    X[5I-4] = (X[5I-1]-X[5I-2]) * X[5I-3];
```
Solution Steps

- Construct an Array Transition Graph (ATG)
  - Compiler analysis
  - Profiling

- Determine important paths
  - (paths that contain edges with high transition counts)

- Interleave the arrays in each path
Discovering Important Paths

- Important Paths: paths with large cumulative weight
- Can be multiple important paths (a path cover)
- Similar to Kruskal’s Spanning Tree Algorithm
- Can be solved in $O(|E|\log|E|)$
- Alternative solutions are possible
Extending the Scope

- Arrays with
  - Same access frequency
  - Same dimensionality
  - Same extents up to a permutation

- Arrays with
  - Same access frequency
  - Same dimensionality
  - Different extents

- Arrays with
  - Same access frequency
  - Different dimensionality
Experiments - I

Energy Consumption (J)
Experiments - II

![Energy Consumption Graph](chart.png)
Experiments - III

Normalized Energy Consumption

- conv
- biquad
- fir
- lms
- complex
- real
- fft
- eflux

Legend:
- 1x8MB Opt
- 2x4MB Opt
- 4x2MB Opt
- 8x1MB No Opt
Experiments - IV

Performance Penalty (%)

- conv
- biquad
- fir
- lms
- complex
- real
- fft
- eflux

4x16KB | 4x32KB | 4x64KB | 4x128KB