Data Locality Enhancement by Memory Reduction

Y. Song, R. Xu, C. Wang, Z. Li

Presented by

Zhiyuan Li

Department of Computer Science

Purdue University

For 15th ACM International Conference on Supercomputing

Talk Outline

- Describe the problem and the basic idea.
- Briefly describe compiler schemes.
- Experimental results.
- (Briefly explain the math framework for minimization.)
- Summary

Memory Reduction for Locality Enhancement

- General idea: Reducing the foot-print to fit in fast memory devices.
- Array contraction is the particular technique here.
  - Full contraction to scalars: convert memory references to register references.
  - Partial contraction to low-dimensional arrays: potentially more reusable cached data.
  - Reduced pressure on the cache, leading to potentially better reusability for noncontracted variables as well.

Main Contributions

- Experiments showing that array contraction can get significant memory performance improvement and speedup.

- Minimizing data size for more general program model.
  - Fraboulet *et al.* (1999) for perfect loop nest only.
  - We perform *loop shifting*, *loop fusion*, *loop interchange*, *etc.* to create more contraction opportunities.

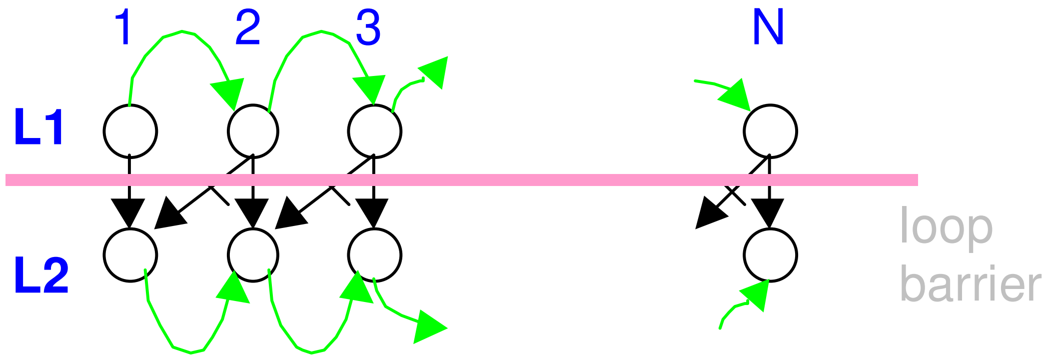
An Example of Full Contraction

```
DO T = 1, ITMAX
L1: DO I = 1, N
        A(I) = E(I) + E(I - 1)
    END DO
L2: DO I = 1, N
        E(I) = A(I)
    END DO
    END DO
```

```
DO T = 1, ITMAX
DO I = 1, N
  A(I) = E(I) + E(I - 1)
END DO
DO I = 2, N + 1
  E(I - 1) = A(I - 1)
END DO
END DO
```

A[1:N] is the *local array region*.

E[0:N] is the *input array region*, and E[1:N] is the *output array region*.

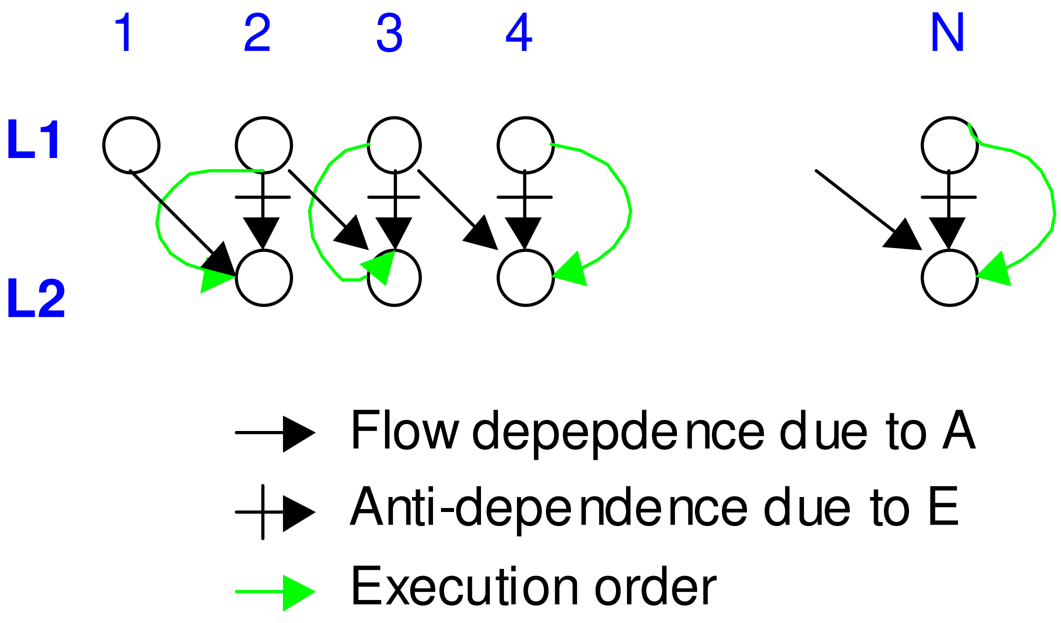


→ Flow dependence due to A

+→ Anti-dependence due to E

→ Execution order

***N* A-elements need to be simultaneously live**



After fusion, only two A-elements are simultaneously live

An Example of Full Contraction (Cont'd)

DO $T = 1, ITMAX$
DO $I = 1, N+1$
  IF ($I.EQ.1$) THEN
    $A(I) = E(I) + E(I-1)$
  ELSE IF ($I.EQ.(N+1)$) THEN
    $E(I-1) = A(I-1)$
  ELSE
    $A(I) = E(I) + E(I-1)$
    $E(I-1) = A(I-1)$
  END IF
END DO
END DO

DO $T = 1, ITMAX$
$a2 = E(1) + E(0)$
DO $I = 2, N$
  $a1 = a2$
  $a2 = E(I) + E(I-1)$
  $E(I-1) = a1$
END DO
$E(N) = a2$
END DO

Benefits Shown in the Last Example

- Convert 50% of the memory references to register references (requiring two registers only).
- Reduce the footprint by half, making the array E elements more reusable in the cache.
  - E.g. suppose the cache is of size C and suppose each original array is also of size C,

An Example of Partial Reduction (Based on LL18)

L1: DO $K = 2, KN$
DO $J = 2, JN$
$ZA(J, K) = ZP(J-1, K+1)$
$+ ZR(J-1, K-1)$
END DO
END DO
L2: DO $K = 2, KN$
DO $J = 2, JN$
$ZB(J, K) = ZQ(J-1, K) + ZZ(J, K)$
END DO
END DO

/* continued */
L3: DO $K = 2, KN$
DO $J = 2, JN$
$ZP(J, K) = ZP(J, K) + ZA(J, K)$
$-ZA(J-1, K) - ZB(J, K)$
$+ ZB(J, K+1)$
END DO
END DO
L4: DO $K = 2, KN$
DO $J = 2, JN$
$ZQ(J, K) = ZQ(J, K) + ZA(J, K)$
$+ZA(J-1, K) + ZB(J, K)$
$+ ZB(J, K+1)$
END DO
END DO

Contract local regions:

ZA[2:JN,2:KN] $\longrightarrow$ $za0$, $za1$. ZB[2:JN,2:KN] $\longrightarrow$ $ZB1[2:ZN]$.

Reshape exposed regions:

ZA[1,2:KN] $\longrightarrow$ $ZA[2:KN]$. ZB[2:JN,KN+1] $\longrightarrow$ $ZB0[2:KN]$.

LL18 Example (Cont'd)

REAL*8 $ZA(2:KN), ZB0(2:JN), za0, za1, ZB1(2:JN), zb$

```
DO J = 2, JN
   ZB1(J) = ZQ(J-1,2) + ZZ(J,2)
END DO
DO K = 3, KN
S1: za1 = ZA(K-1)
   DO J = 2, JN
S2: za0 = za1
      za1 = ZP(J-1,K) + ZR(J-1,K-2)
      zb = ZQ(J-1,K) + ZZ(J,K)
      ZP(J,K-1) = ZP(J,K-1) + za1 - za0
         -ZB1(J) + zb
      ZQ(J,K-1) = ZQ(J,K-1) + za1 + za0
         +ZB1(J) + zb
S3: ZB1(J) = zb
   END DO
END DO
```

```
S4: za1 = ZA(KN)
DO J = 2, JN
S5: za0 = za1
   za1 = ZP(J-1,KN+1)
                   + ZR(J-1,KN-1)
   ZP(J,KN) = ZP(J,KN)+za1 - za0
     -ZB1(J) + ZB0(J)
   ZQ(J,KN) = ZQ(J,KN)+za1 + za0
      ZB1(J) + ZB0(J)
END DO
```

Benefits Shown in the Last Example

- Most of the memory references to array ZA are converted to register references (requiring two registers only). (Memory reference counts reduced by about 35%.)
- Most of the references to two-dimensional array ZB are converted to references to a single-dimensional array — improved reusability.
- Overall footprint is reduced.
- Experiments show that L2 cache misses are reduced by about 50%. Execution time reduced by about 45%.

The Sequence of Transformations

The Original Loop Nest

L_1 : **DO** $L_{1,1} = l_{1,1}, u_{1,1}$
  DO $L_{1,2} = l_{1,2}, u_{1,2}$
    $\ldots$
      DO $L_{1,n} = l_{1,n}, u_{1,n}$
$\ldots$
L_i : **DO** $L_{i,1} = l_{i,1}, u_{i,1}$
  DO $L_{i,2} = l_{i,2}, u_{i,2}$
    $\ldots$
      DO $L_{i,n} = l_{i,n}, u_{i,n}$
$\ldots$
L_m : **DO** $L_{m,1} = l_{m,1}, u_{m,1}$
  DO $L_{m,2} = l_{m,2}, u_{m,2}$
    $\ldots$
      DO $L_{m,n} = l_{m,n}, u_{m,n}$

- **To conform to the loop model, may insert dummy loop levels and guards.**
- **May also interchange loops to match loop levels according to array indices.**

After Shifting

L_1 : **DO** $L_{1,1} = l_{1,1} + p^1(L_1), u_{1,1} + p^1(L_1)$
DO $L_{1,2} = l_{1,2} + p^2(L_1), u_{1,2} + p^2(L_1)$
$\ldots$
DO $L_{1,n} = l_{1,n} + p^n(L_1), u_{1,n} + p^n(L_1)$
$\ldots$
L_i : **DO** $L_{i,1} = l_{i,1} + p^1(L_i), u_{i,1} + p^1(L_i)$
DO $L_{i,2} = l_{i,2} + p^2(L_i), u_{i,2} + p^2(L_i)$
$\ldots$
DO $L_{i,n} = l_{i,n} + p^n(L_i), u_{i,n} + p^n(L_i)$
$\ldots$
L_m : **DO** $L_{m,1} = l_{m,1} + p^1(L_m), u_{m,1} + p^1(L_m)$
DO $L_{m,2} = l_{m,2} + p^2(L_m), u_{m,2} + p^2(L_m)$
$\ldots$
DO $L_{m,n} = l_{m,n} + p^n(L_m), u_{m,n} + p^n(L_m)$

- **To make the next fusion step legal, each data dependence between two loops places a lower bound on the shifting factor.**
- **To minimize the temporaries, the shifting factor is to be minimized, subject to data dependences.**
- **This becomes an optimization problem.**

After Fusion w/o Coalescing

DO $J_1 = min_{i=1}^{m}(l_{i,1} + p^1(L_i)),\ max_{i=1}^{m}(u_{i,1} + p^1(L_i))$

  DO $J_2 = min_{i=1}^{m}(l_{i,2} + p^2(L_i)),\ max_{i=1}^{m}(u_{i,2} + p^2(L_i))$

    $\ldots$

      DO $J_n = min_{i=1}^{m}(l_{i,n} + p^n(L_i)),\ max_{i=1}^{m}(u_{i,n} + p^n(L_i))$

- Theorem: Minimizing memory required for the given loop nests is equivalent to:

  determining the shifting vector $p(L_j)$ for each nested loop L_j in the loop collection, s.t. the fused loop has the minimum numer of simultaneously live array elements.

Coalescing + Fusion

After Coalescing

DO $L_1 = l_1, u_1$

$\quad \ldots$

DO $L_i = l_i, u_i$

$\quad \ldots$

DO $L_m = l_m, u_m$

$\quad \ldots$

After Fusion

DO $L = min_{i=1}^{m} l_i, max_{i=1}^{m} u_i$

$\quad \ldots$

- **Loop coalescing to ensure minimal memory-size in general cases.**
- **Loop coalescing is unnecessary in most common cases.**

Overall Compiler Scheme

- Transform program to conform to the loop model.
- Determine shifting vectors to minimize memory size after fusion, using the Successive Shortest Path Algorithm on a network flow problem.
- Perform "controlled fusion" (Procedure Pick_and_Reject) to prevent register spills and cache overflow.

Test Programs for Experiments

LL14	purdue-02
LL18	purdue-03
Jacobi	purdue-04
swim95 (SPEC95fp)	purdue-07
swim00 (SPEC2000fp)	purdue-08
tomcatv (SPEC95fp)	purdue-12
hydro2d (SPEC95fp)	purdue-13
lucas (SPEC2000fp)	climate (Rice U.)
mg (NPB2.3-serial)	laplace-jb (Rice U.)
combustion (UMD Chaos group)	laplace-gs (Rice U.)

Summary

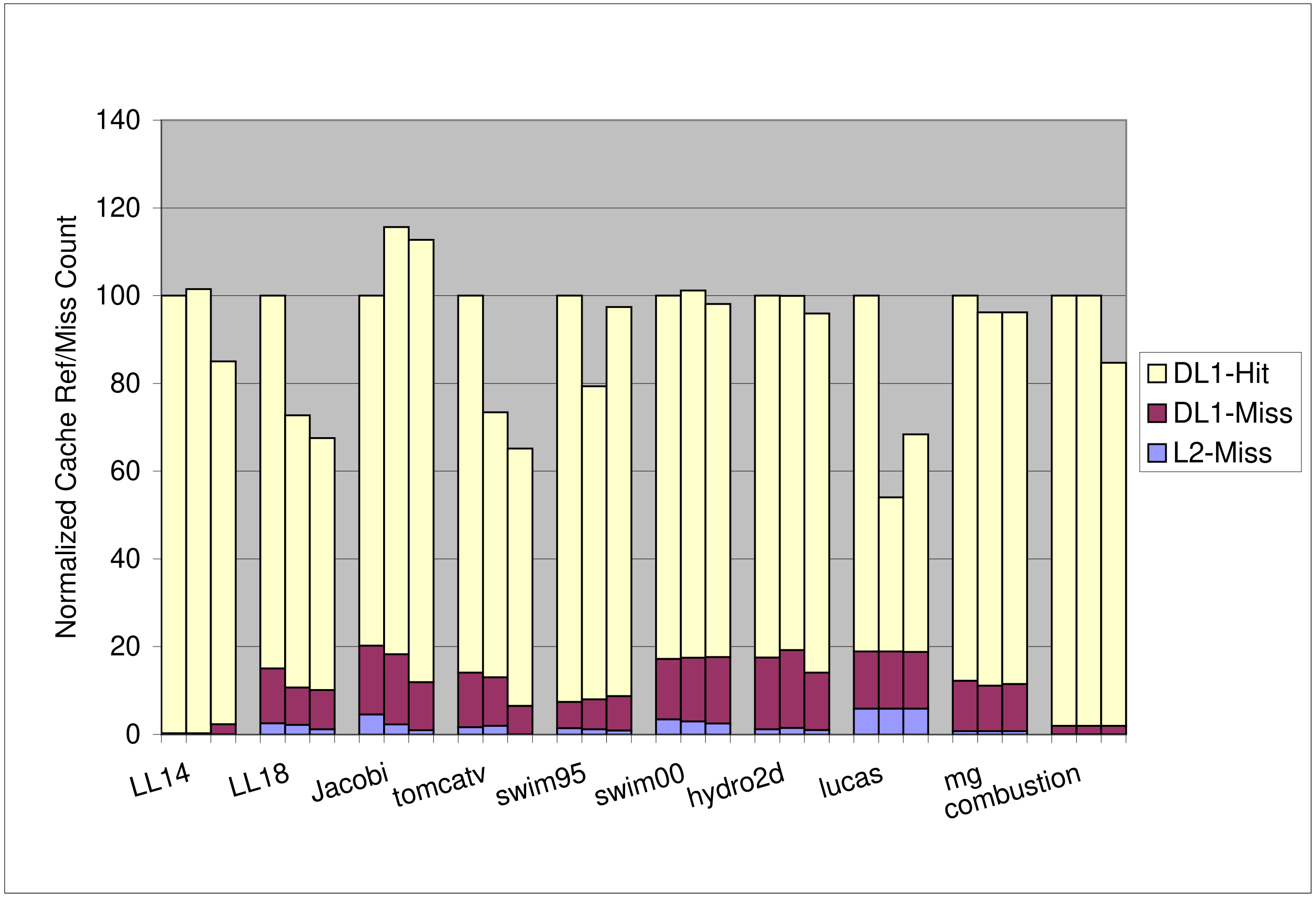
- Extend *array contraction* technique by allowing partial reduction and more general loop nesting.
- A polynomial-time ($O(\mid V \mid^3)$) algorithm to minimize memory requirement for shifting, fusion and contraction.
- A simple heuristic to control fusion.
- Where array contraction is applicable, result can be substantial: memory reference counts are reduced by 21.1% overall, cache misses are reduced by 63.8% overall. Speeding up program execution by a factor of 1.4 on average.

Acknowledgements

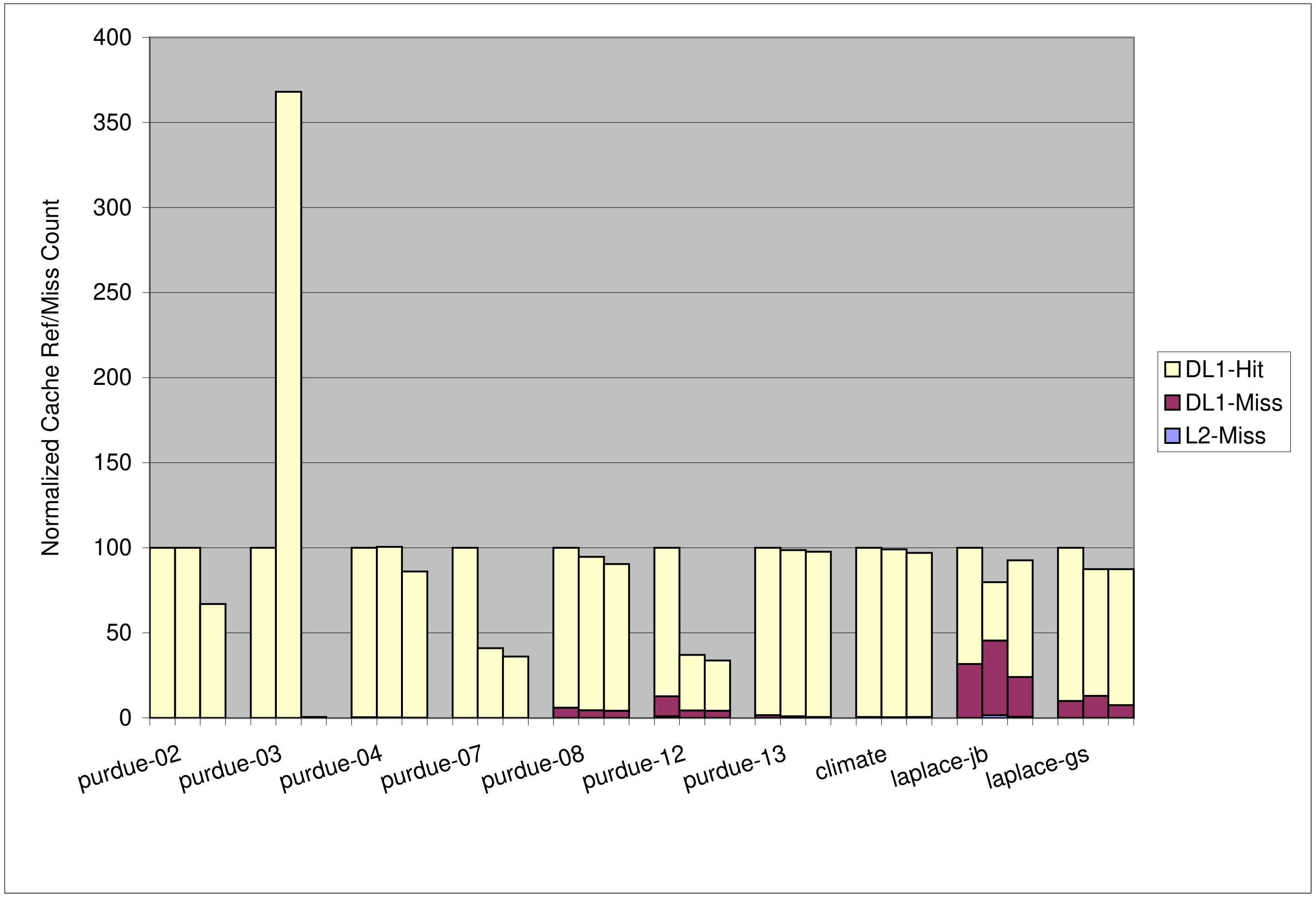
This work is sponsored in part by

- National Science Foundation through grants CCR-9975309, ACI/ITR-0082834 and MIP-9610379,
- by Indiana 21st Century Fund,
- by Purdue Research Foundation,
- and by a donation from Sun Microsystems, Inc.

(Memory behavior on R10K: Original, fusion-only and fusion+contraction are from left to right)



(Memory behavior on R10K (cont'd): Original, fusion-only and fusion+contraction are from left to right)



(Execution time on SGI/R10K: Original, fusion-only and fusion+contraction are from left to right)