A unified compiler algorithm for optimizing locality, parallelism, and communication in out-of-core computations

M. Kandemir

A. Choudhary

J. Ramanujam

M. Kandaswamy

Syracuse, Northwestern, Louisiana State University

Fifth Workshop on I/O in Parallel and Distributed Systems
November, 1997

Outline

- optimizing for file locality
- optimizing for parallelism
- optimizing for file locality + parallelism
- preliminary results
- summary + work-in-progress

Optimizing for Locality

loop transformations:

- linear loop transformations for optimizing locality
- tiling (blocking) *mandatory for out-of-core computations*

data transformations:

- changing layout of data in memory (e.g. re-indexing dimensions)
- customizing data layout (e.g. padding)

unified transformations:

- applying loop and data transformations in concert

Optimizing for Locality

goals:

- determine the appropriate file layouts for each array
- determine accompanying loop transformations
- determine optimal memory allocation across arrays

Example: Matrix Multiplication

original code:

$$
\begin{array}{l}
\textbf{do } i = 1, N \\
\quad \textbf{do } j = 1, N \\
\quad\quad \textbf{do } k = 1, N \\
\quad\quad\quad C(i,j) += A(i,k) + B(k,j) \\
\quad\quad \textbf{enddo} \\
\quad \textbf{enddo} \\
\textbf{enddo}
\end{array}
$$

all arrays are out-of-core and reside on disk. *data reference matrices:*

$$C_{\mathcal{R}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}; A_{\mathcal{R}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}; B_{\mathcal{R}} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Example –continued

optimized code (*loop trans.*):

do $j = 1, N$

  do $k = 1, N$

    do $i = 1, N$

      $C(i, j) += A(i, k) + B(k, j)$

    enddo

  enddo

enddo

optimized code (*data trans.*–bold letters are row-major):

do $i = 1, N$

  do $j = 1, N$

    do $k = 1, N$

      $\mathbf{C}(i, j) += \mathbf{A}(i, k) + B(k, j)$

    enddo

  enddo

enddo

Strategy

On applying an invertible transformation T, the transformed data reference matrix:

$$A'_{\mathcal{R}} = A_{\mathcal{R}} T^{-1}$$

Source loop nest : $(i_1, i_2, \ldots, i_n)$
Transformed loop nest : $(j_1, j_2, \ldots, j_n)$

$$T : \begin{pmatrix} i_1 \\ \vdots \\ i_n \end{pmatrix} \longrightarrow \begin{pmatrix} j_1 \\ \vdots \\ j_n \end{pmatrix}$$

Columns of T^{-1} determine subscripts of array references in transformed loop nest

Use reference matrices to construct the best T^{-1}

Algorithm for Locality

a three-deep loop nest where two dimensional arrays are accessed

$$A_{\mathcal{R}} = \begin{pmatrix} \times & \times & \times \\ \times & \times & \times \end{pmatrix}; T^{-1} = \begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix}$$

for each reference (starting from the LHS) the following order is tried

- $A_{\mathcal{R}}T^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ \times & \times & 0 \end{pmatrix}$ or $A_{\mathcal{R}}T^{-1} = \begin{pmatrix} \times & \times & 0 \\ 0 & 0 & 1 \end{pmatrix}$ (*best*)
- $A_{\mathcal{R}}T^{-1} = \begin{pmatrix} 0 & 1 & 0 \\ \times & 0 & 0 \end{pmatrix}$ or $A_{\mathcal{R}}T^{-1} = \begin{pmatrix} \times & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ (*less optimized*)

Example: Matrix Multiplication revisited

do $i = 1, N$
    do $j = 1, N$
        do $k = 1, N$
            $\mathbf{C}(i, j) += \mathbf{A}(i, k) + B(k, j)$
        enddo
    enddo
enddo

$$C_{\mathcal{R}}T^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} (less\ optimized);$$

$$A_{\mathcal{R}}T^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} (best);\ B_{\mathcal{R}}T^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} (best).$$

Optimizing for Parallelism

goals:

- distribute the outermost iterations of the transformed loop nest
- distribute *a* corresponding dimension of the LHS array
- move communication due to an RHS references out of the innermost loop
- use block transfers (*message vectorization*)

Algorithm for Parallelism

a three-deep loop nest where two dimensional arrays are accessed

$$A_{\mathcal{R}} = \begin{pmatrix} \times & \times & \times \\ \times & \times & \times \end{pmatrix}; T^{-1} = \begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix}$$

- *for the LHS reference and compatible RHS references:*
  $A_{\mathcal{R}}T^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ \times & \times & \times \end{pmatrix}$ or $A_{\mathcal{R}}T^{-1} = \begin{pmatrix} \times & \times & \times \\ 1 & 0 & 0 \end{pmatrix}$ (*best*)

- *for the remaining RHS references:* $A_{\mathcal{R}}T^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ \times & \times & 0 \end{pmatrix}$ or

  $A_{\mathcal{R}}T^{-1} = \begin{pmatrix} \times & \times & 0 \\ 0 & 0 & 1 \end{pmatrix}$ (*less optimized*)

Example: SYR2K

do $i = 1, N$
    do $j = i, \min(i + 2b - 2, N)$
        do $k = \max(i - b + 1, j - b + 1, 1), \min(i + b - 1, j + b - 1, N)$
            $C(i, j - i + 1) \mathrel{+}= A(k, i - k + b) * B(k, j - k + b)$
                $+ A(k, j - k + b) * B(k, i - k + b)$
        enddo
    enddo
enddo

$$C_{\mathcal{R}} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \end{pmatrix} \qquad A^1_{\mathcal{R}} = B^2_{\mathcal{R}} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & -1 \end{pmatrix}$$

$$A^2_{\mathcal{R}} = B^1_{\mathcal{R}} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & -1 \end{pmatrix}$$

SYR2K – LHS row distribution

$$C_{\mathcal{R}}T^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ \times & \times & \times \end{pmatrix}$$

<u>Derive distribution of A</u>

$A^1_{\mathcal{R}}T^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ \times & \times & 0 \end{pmatrix}$ is impossible; $A^1_{\mathcal{R}}T^{-1} = \begin{pmatrix} \times & \times & 0 \\ 0 & 0 & 1 \end{pmatrix}$ is impossible

$A^2_{\mathcal{R}}T^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ \times & \times & 0 \end{pmatrix}$ is impossible; $A^2_{\mathcal{R}}T^{-1} = \begin{pmatrix} \times & \times & 0 \\ 0 & 0 & 1 \end{pmatrix}$ is impossible

<u>Derive distribution of B</u>

Similar analysis. There is no way to distribute arrays A and B (when the first dimension of C is distributed) to allow block transfers outside innermost loop

SYR2K – LHS column distribution

$$C_{\mathcal{R}} T^{-1} = \begin{pmatrix} \times & \times & \times \\ 1 & 0 & 0 \end{pmatrix}$$

<u>Derive distribution of A</u>

$A^1_{\mathcal{R}} T^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ \times & \times & 0 \end{pmatrix}$ is OK; $A^2_{\mathcal{R}} T^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ \times & \times & 0 \end{pmatrix}$ is OK

<u>Derive distribution of B</u> Similar analysis.

One possible solution: $T^{-1} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \rightarrow T = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix}.$

SYR2K – Final code

do $u = 1, N$

  do $v = u, \min(u + 2b - 2, N)$

    receive $A(:, v + b)$

    receive $A(:, u + v + b)$

    receive $B(:, u + v + b)$

    receive $B(:, v + b)$

    do $w = \max(u - b + 1, v - b + 1, 1), \min(u + b - 1, v + b - 1, N)$

      $C(v + w + 1, u) \mathrel{+}= A(w, v + b) * B(w, u + v + b)$

      $\qquad + A(w, u + v + b) * B(w, v + b)$

    enddo

  enddo

enddo

Iterations of u loop distributed among processors!

Optimizing for Locality and Parallelism

goals:

- find optimal file layouts
- distribute the outermost iterations of the transformed loop nest
- use block transfers (*message vectorization*)

approach: try to find optimal data reference matrices that satisfy "file locality" *and* "parallelism" requirements

Algorithm for Locality and Parallelism

a three-deep loop nest where two dimensional arrays are accessed

$$A_{\mathcal{R}} = \begin{pmatrix} \times & \times & \times \\ \times & \times & \times \end{pmatrix}; T^{-1} = \begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix}$$

alternatives:

$$A_{\mathcal{R}}T^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \text{ or } A_{\mathcal{R}}T^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \ (\textit{best})$$

$$A_{\mathcal{R}}T^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ \times & 1 & 0 \end{pmatrix} \text{ or } A_{\mathcal{R}}T^{-1} = \begin{pmatrix} \times & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \ (\textit{less optimized})$$

$$A_{\mathcal{R}}T^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ \times & \times & 0 \end{pmatrix} \text{ or } A_{\mathcal{R}}T^{-1} = \begin{pmatrix} \times & \times & 0 \\ 0 & 0 & 1 \end{pmatrix} \ (\textit{less optimized})$$

Global Locality Optimization

The impact of layout optimization is global. This can be mitigated by loop transformations. The general problem is NP-complete!

1st approach:

- each loop is optimized for the best locality
- a cost criterion is used to choose the global layouts

2nd approach:

- loop nests are ordered according to a *cost* criteria
- as the nests are optimized the layouts are propagated

Example: A synthetic benchmark

$$
\begin{array}{l}
\textbf{do } i = 1, N \\
\quad \textbf{do } j = 1, N \\
\qquad A(i,j) = B(j,i) * C(i,j) + D(i,j) * \textbf{LOG}(E(j,i)) \\
\quad \textbf{enddo} \\
\textbf{enddo}
\end{array}
$$

$$
\begin{array}{l}
\textbf{do } i = 1, N \\
\quad \textbf{do } j = 1, N \\
\qquad B(i,j) = A(j,i) + E(i,j) \\
\quad \textbf{enddo} \\
\textbf{enddo}
\end{array}
$$

default: all arrays are column-major

Example: A synthetic benchmark

$$
\begin{array}{l}
\textbf{do } j = 1, N \\
\quad \textbf{do } i = 1, N \\
\quad\quad A(i,j) = B(j,i) * C(i,j) + D(i,j) * \textbf{LOG}(E(j,i)) \\
\quad \textbf{enddo} \\
\textbf{enddo}
\end{array}
$$

$$
\begin{array}{l}
\textbf{do } i = 1, N \\
\quad \textbf{do } j = 1, N \\
\quad\quad B(i,j) = A(j,i) + E(i,j) \\
\quad \textbf{enddo} \\
\textbf{enddo}
\end{array}
$$

A, C and D are column-major; B and E are row-major

A four-deep nest

original code:

do $i = 1, N$
    do $j = 1, N$
        do $k = 1, N$
            do $l = 1, N$
                $A(i,j) += B(k,i) + C(l,k)$
            enddo
        enddo
    enddo
enddo

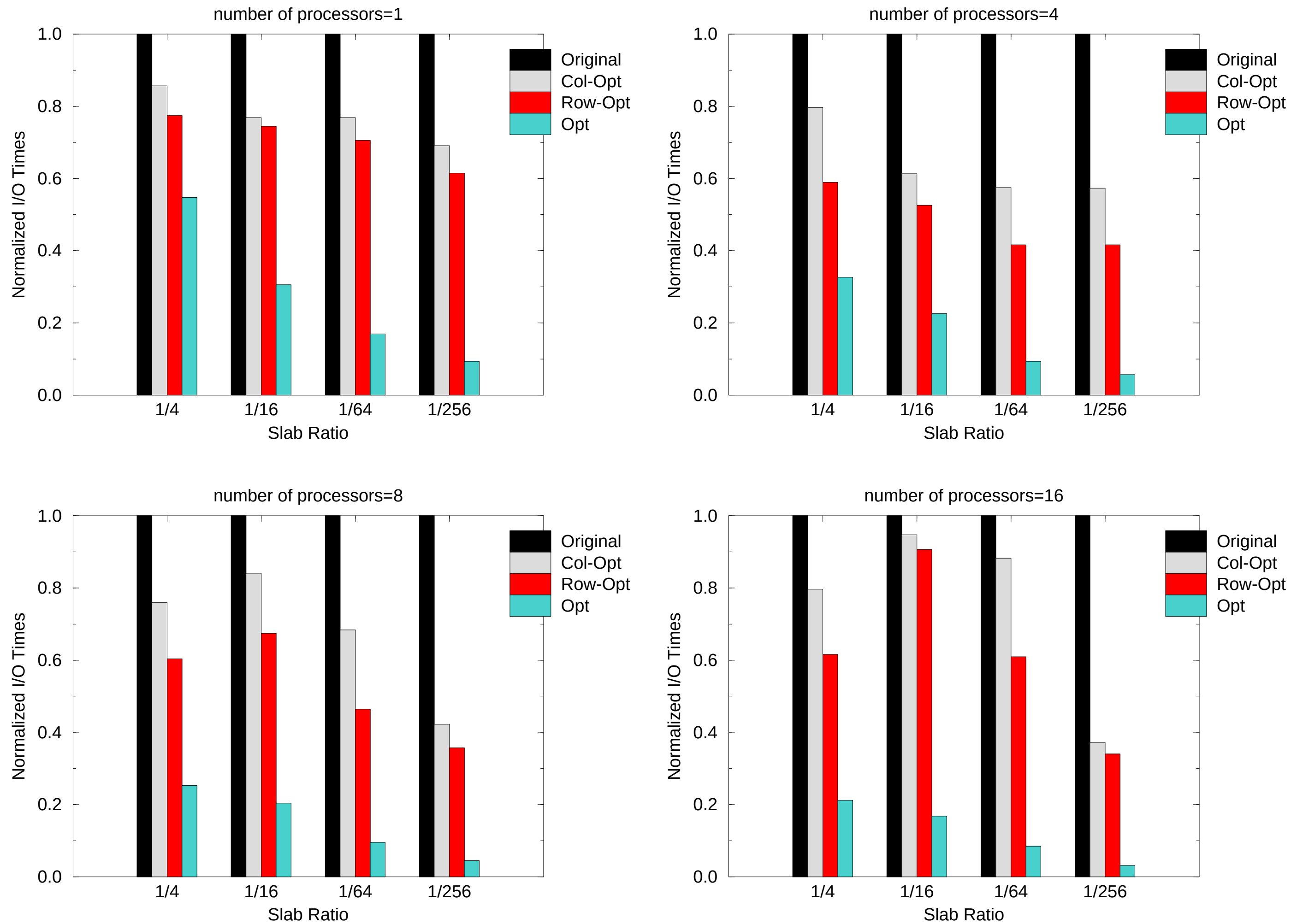
I/O times on SP-2 and Paragon

	IBM SP-2							
	$2K \times 2K$ double arrays				$4K \times 4K$ double arrays			
SR	Ori	Col	Row	Opt	Ori	Col	Row	Opt
1/4	152	138	131	77	363	311	281	199
1/16	623	577	524	105	1,441	1,108	1,074	441
1/64	4,224	3,111	2,462	563	8,384	6,449	5,912	1,422
1/256	25,087	21,627	19,298	1,566	48,880	35,759	30,055	4,584

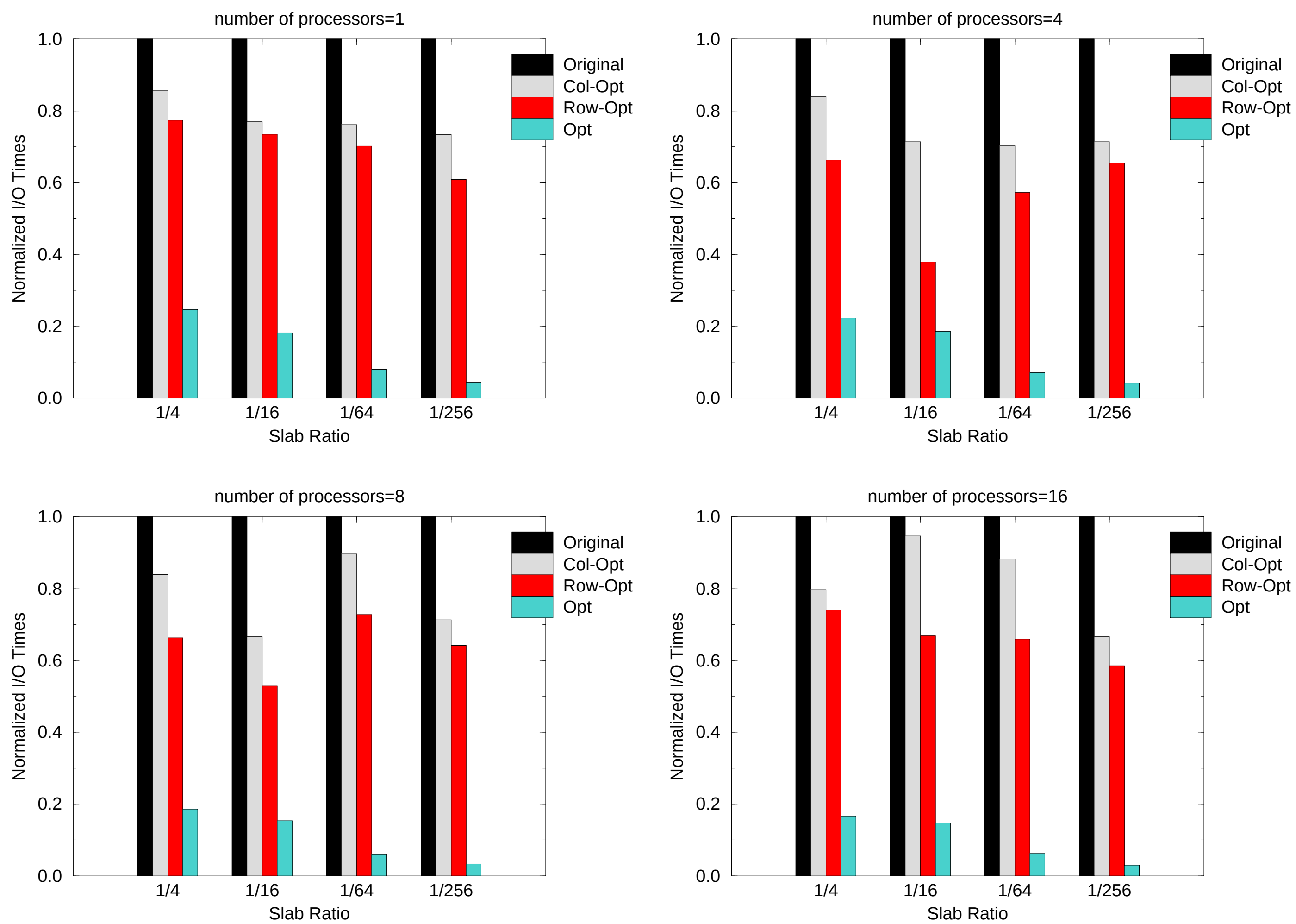
	Intel Paragon							
	$2K \times 2K$ double arrays				$4K \times 4K$ double arrays			
SR	Ori	Col	Row	Opt	Ori	Col	Row	Opt
1/4	1,159	1,050	988	208	5,172	4,431	4,001	1,273
1/16	2,320	2,141	1,951	252	7,360	5,668	5,412	1,344
1/64	19,201	14,141	11,760	576	28,864	22,002	20,257	2,304
1/256	99,583	83,848	76,245	3,028	192,512	141,370	117,270	8,193

"SR" is a measure of available memory.

Normalized I/O times on SP-2



Normalized I/O times on Paragon



Observations

- optimized programs have much less execution times
- for a fixed SR, when p increased, the effectiveness increases
- for a fixed p, when SR is decreased, the effectiveness increases
- optimized programs scales better
- the results are conservative

Summary

- dense and band matrix codes
- spatial locality in files is exploited
- one dimension of an array is distributed
- find a good combination of data layout and data distribution

In progress

- more than one distributed dimension
- reduce complexity of search
- relax owner-compute
- experimenting with large applications and better global algorithms