Compiler Algorithms for Optimizing Locality and Parallelism on Shared and Distributed Memory Machines

M. Kandemir

J. Ramanujam

A. Choudhary

Syracuse, Louisiana State, Northwestern University

Parallel Architectures and Compilation Techniques (PACT)
November, 1997

Outline

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

Optimizing for Locality

loop transformations:

- linear loop transformations for optimizing locality
- tiling (blocking)

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

Example: Matrix Multiplication

original code:

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

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

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

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

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

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

Derive distribution of A

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

Derive distribution of B

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: $T^{-1} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \to T = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix}.$

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 $w = \max(u - b + 1, v - b + 1, 1), \min(u + b - 1, v + b - 1, N)$
      $C(v + w + 1, u) \mathrel{+}= A(w, v + b) * B(w, u + v + b)$
        $+ A(w, u + v + b) * B(w, v + b)$
    enddo
  enddo
enddo

Iterations of u loop distributed among processors!

Optimizing for Locality and Parallelism

goals:

- find optimal data 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 "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 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

Preliminary Results for MxM on Convex Exemplar

execution times (in seconds) when N=512:

`version`	proc = 8	proc = 16
original	`29.817`	`14.412`
jki-version	`4.508`	`3.487`
layout (ijk)	`3.156`	`3.288`

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

Preliminary Results on Convex Exemplar

execution times (in seconds) when N=64:

`version`	proc = 8	proc = 16
original	19.086	7.542
ikjl-version	18.225	7.273
jkli-version	20.140	7.276
layout (A is row-major)	14.280	7.056

Observations

Our optimizations

- reduce miss rates and execution times
- improve performance of tiled code
- reduce sensitivity of tiling to the tile size
- are successful in eliminating false sharing (not evaluated yet)

Summary

- dense and band matrix codes
- spatial locality is optimized
- 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
- application of algorithm to I/O based (e.g. out-of-core) codes