Principles for balancing data locality and coarse-grained parallelism in loop permutation

Guansong Zhang, Roch Archambault, Raul Silvera, Peng Zhao

IBM Toronto Lab
Toronto
ON, L6G 1C7, Canada

Abstract. In this note, we present some principles for loop selection algorithm for permuting nested loops in an auto-parallelizing compiler. It will help us to further develop algorithms considering both data locality and coarse-grained parallelism.

Keywords: parallelizing compiler, data dependence, loop permutation, data locality, parallel do

1 Description

Before we presenting our discussion in details, we need to introduce the underlining model to analyze the problem.

1.1 Analyzing model

The data dependence vectors[1] are still our principle tools to analyze and verify the transformation that we are going to apply on a loop nest. Suppose $\boldsymbol{\delta} = \{\delta_1 \ldots \delta_n\}$ is a hybrid distance/direction vector with the most precise information derivable. It represents a *data dependence* between two array references, corresponding left to right from the outermost loop to innermost loop enclosing the references. Data dependences are *loop-independent* if the accesses to the same memory location occur in the same loop iteration; they are *loop-carried* if the accesses occur on different loop iterations.

For example, if we have

```
    DO i_1 = 1, U_1
      DO i_2 = 1, U_2
        ...
        DO i_n = 1, U_n
S_1       A(f_1(i_1,...,i_n),...,f_m(i_1,...,i_n))=...
S_2       ...=A(g_1(i_1,...,i_n),...,g_m(i_1,...,i_n))
T_1       B(u_1(i_1,...,i_n),...,u_m(i_1,...,i_n))=...
T_2       ...=B(v_1(i_1,...,i_n),...,v_m(i_1,...,i_n))
        END DO
        ...
      END DO
    END DO
```

as a loop nest from L_1 to L_n, we may have two dependences $\boldsymbol{\delta}_A$ and $\boldsymbol{\delta}_B$ characterizing the nested loops, which were caused by S_1/S_2 and T_1/T_2. For simplicity, we assume that f, g, u, v are linear functions of the loop induction variables in this note.

Furthermore,

$$\begin{bmatrix} \boldsymbol{\delta}_A \\ \boldsymbol{\delta}_B \end{bmatrix}$$

is a $2 \times n$ *dependence matrix*, denoted as $\mathcal{D}$. More generally, the number of rows in a dependence matrix may be from one to any positive number, depending on how many references in the loop body.

We introduce two theorems.

Theorem 1 (C-level interchangeable). *Loop L_c and L_{c-1} is interchangeable if and only if that $\forall \boldsymbol{\delta} \in \mathcal{D}$, $\boldsymbol{\delta} \neq (=^{(c-2)}<> \ldots)$, where "$=^{(c-2)}$" means there are $(c-2)$ directions as "$=$" before the first "$<$".*

Theorem 1 and its variant forms can be found in [2] and other literatures. We will not prove it here, and use it directly for our own theorem later.

Theorem 2 (P-level parallelizable). *Loop L_p can be parallelized as a parallel do if and only if that $\forall \boldsymbol{\delta} \in \mathcal{D}$, $\boldsymbol{\delta} \neq (=^{(p-1)}< \ldots)$.*

The proof for Theorem 2 is trivial. By the definition of parallel do, it can not carry any dependence.

Next, we define a *multiply* operation on a dependence vector.

Let $\boldsymbol{\sigma} = \boldsymbol{\sigma}_a \times \boldsymbol{\sigma}_b$, where $\forall \sigma_j \in \boldsymbol{\sigma}, \sigma_j = \sigma_{a_j} \times \sigma_{b_j}$. The multiplication of the two dependence distance is defined as following, it is unknown if one of the distance is unknown. Otherwise it will be the regular multiplication on two integers. If one of the distance is only a direction, it is treated as 1 or -1, and after the multiplication, converted back to a direction .

Let $\boldsymbol{\sigma}_p$ and $\boldsymbol{\sigma}_{p-1}$ be the p^{th}, $(p-1)^{th}$ column vector of dependence matrix $\mathcal{D}$, then we have

Theorem 3 (Keeping parallelizable). *A p-level parallelizable loop L_p can be interchanged to L_{p-1} and becomes (p-1)-level parallelizable if and only if that $\forall \sigma_j \in \boldsymbol{\sigma}$, where $\boldsymbol{\sigma} = \boldsymbol{\sigma}_{p-1} \times \boldsymbol{\sigma}_p$, either*

- *$\sigma_j = 0$ or*
- *the j^{th} dependence in $\mathcal{D}$ is not carried by L_{p-1}*

Prof

We start from proving the "*if*" direction of the theorem.

Suppose $\forall \sigma_j = 0$.

Fist we can assert that L_{p-1} and L_p is interchangeable. Otherwise, according to Theorem 1 there is a dependence vector $\boldsymbol{\delta}$ has the form of $(=^{(p-2)}<> \ldots)$. This will conflict with the fact that all σ_j is zero. Second, we prove that the L_p will become $(p-1)$-level parallelizable after interchanging with L_{p-1}. Otherwise, from Theorem 2, there will a dependence vector $\boldsymbol{\delta}$, of the format $(=^{(p-2)}< \ldots)$, in the dependence matrix preventing it from being parallelized after the interchange. Considering the format of the $\boldsymbol{\delta}$ before the interchange, given that all σ_j is zero, it should be in the form of $(=^{(p-1)}< \ldots)$. This conflicts the fact that L_p is parallelizable according to Theorem 2.

If $\exists\sigma_j \neq 0$, we let $\boldsymbol{\delta}$ be the dependence vector on the j^{th} row of the matrix. It should have one of the following formats $(\cdots^{(p-2)} << \ldots)$, $(\cdots^{(p-2)} <> \ldots)$, $(\cdots^{(p-2)} >< \ldots)$, $(\cdots^{(p-2)} >> \ldots)$, where $\cdots^{(p-2)}$ is the leading $(p-2)$ positions. Since L_{p-1} does not carry any dependences as in the given condition, $\cdots^{(p-2)}$ has be in the format of $(= \ldots =< \ldots)$, in order for the δ to be valid. Therefor L_p can be interchanged with L_{p-1} and kept being parallelizable.

Similarly we can prove the "*only if*" direction. It is not important in our algorithm, we will omit it here.

End of Prof

Theorem 4 (Outermost parallelizable). *Loop L_o can be parallelized at the outermost level if and only if that $\boldsymbol{\sigma} = \mathbf{0}$, where $\boldsymbol{\sigma}$ is the o^{th} column vector of dependence matrix $\mathcal{D}$.*

This is a direct conclusion from Theorem 3. It is actually used in [3] for loop selection.

2 Summary

Theorem 3 provides a simple way to check whether the loop interchanged is still legal to be parallelized. We can further use it to develop loop-permutation algorithms.

References

1. U. Banerjee. *Dependence Analysis for Supercomputing*. Boston MA: Kluwer, 1988.
2. Hans Zima and Barbara Chapman. *Supercompilers for Parallel and Vector Computers*. Addison-wesley, 1990.
3. Randy Allen and Ken Kennedy. *Optimizing compilers for modern architectures*. Morgan Kaufmann Publishers, 2002.