D Newsletter # 8

NP-completeness of Dynamic Remapping

Ulrich Kremer

CRPC-TR93330-S
August, 1993

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

From the *Proceedings of the Fourth Workshop on Compilers for Parallel Computers*, Delft, The Netherlands, December 1993.

NP-completeness of Dynamic Remapping *

Ulrich Kremer

kremer@cs.rice.edu

Center for Research on Parallel Computation
Department of Computer Science
Rice University
P.O. Box 1892
Houston, Texas 77251

1 Introduction

This paper focuses on the complexity of automatic data layout techniques for regular problems in the context of an advanced compilation system that allows dynamic data remapping. Regular problems allow the compilation system to determine the communication requirements and to perform a variety of program optimizations at compile time. The initial step of the strategy proposed for automatic data layout in the presence of dynamic remapping is to partition the program into code segments, called program *phases*. Phases are intended to represent program segments that perform operations on entire data objects. Data remapping is allowed only between phases. An operational definition of a phase is given elsewhere [KMCKC93]. Strategies for identifying program phases are a topic of current research.

A data layout for a single phase is specified by the alignment and distribution of the arrays referenced in the phase. A data layout for an entire program consists of a collection of data layouts, one data layout for each phase in the program. The data layout for an entire program is determined in three steps. First, alignment analysis builds a search space of reasonable alignment schemes for each phase. Then, distribution analysis uses the alignment search spaces to build candidate data layout search spaces of reasonable alignments and distributions for each phase. A first discussion of possible pruning heuristics and the sizes of their resulting search spaces can be found in [KK93]. Finally, a single candidate data layout scheme for each phase is selected, resulting in a data layout for the entire program. The selection process must consider the tradeoff between the exploitable parallelism of data

*This research was supported by the Center for Research on Parallel Computation (CRPC), a Science and Technology Center funded by NSF through Cooperative Agreement Number CCR-9120008. This work was also sponsored by DARPA under contract #DABT63-92-C-0038, and the IBM corporation. The content of this paper does not necessarily reflect the position or the policy of the U.S. Government and no official endorsement should be inferred.

layouts for each phase and the remapping costs of data layouts between phases. This last step solves the so called *inter-phase* data layout problem. This paper focuses on proving that the inter-phase data layout problem is NP-complete.

Several researchers have published NP-completeness results in the context of automatic data layout. Li and Chen showed that the problem of determining an optimal static alignment between the dimensions of distinct arrays is NP-complete [LC90]. Gilbert and Schreiber proved that aligning temporaries in expression trees involving array sections is NP-complete in the presence of common subexpressions [GS91]. Anderson and Lam showed that the dynamic data layout problem is NP-hard in the presence of control flow between loop nests [AL93]. Mace discussed three different formulations of the dynamic data layout problem for interleaved memory machines. She showed that all three problems are NP-complete [Mac87]. The NP-completeness result presented in this paper is similar to Mace's third formulation of the automatic data layout problem. It turns out that even in the absence of control flow between phases and in the presence of a constant upper bound of the number of candidate data layouts for each phase, the inter-phase data layout problem is NP-complete. The proof reduces the 3-CNF satisfiability problem [CLR90] to the inter-phase data layout problem.

2 Problem Statement

For the purposes of this paper, programs are represented as a linear sequence of phases, $P_1, \ldots P_n$, with no control flow between phases. Let V denote the set of variables in the program, $V = \{v_1, \ldots v_r\}$. Let p_i denote the variables referenced in the i-th phase P_i, i.e. $p_i \subseteq 2^V, 1 \leq i \leq n$. For each P_i there is a set of candidate data layouts $D_i = \{d_i^1, \ldots d_i^{m_i}\}$. Note that two phases may have a different number of candidate data layouts. A single candidate data layout $d_i^k = \{d_{ij_1}^k, \ldots d_{ij_{q_i}}^k\}, 1 \leq k \leq m_i$, is a set of layouts, one layout for each variable $v \in p_i = \{v_{j_1}, \ldots v_{j_{q_i}}\}$.

The cost of executing phase P_i under the data layout $d_i^k \in D_i$ is denoted by $c(P_i, d_i^k)$. The remapping cost from one data layout scheme to another can be defined based on the remapping costs of each individual variable common to both schemes. Let d_α^s and d_β^t be two candidate data layouts for phase P_α and phase P_β, respectively. The remapping cost is given below:

$$c(d_\alpha^s, d_\beta^t) = \sum_{v_i \in p_\alpha \cap p_\beta} c(d_{\alpha i}^s, d_{\beta i}^t),$$

where $c(d_{\alpha i}^s, d_{\beta i}^t)$ is the cost for remapping the single variable v_i.

Let $f_i : p_i \rightarrow \{1, \ldots n\}$ be a mapping that determines for each variable $v \in p_i$ the phase that most recently referenced v before P_i. If no such phase exists, then $f_i(v)$ has the value i. A data remapping of v may occur between phase $P_{f_i(v)}$ and phase P_i.

Definition 1 *An instance of the inter-phase data layout problem consists of a program with a linear sequence of n phases, a set of program variables $V = \{v_1, \ldots v_r\}$, sets p_i and D_i for each phase P_i, and cost functions $c(P_i, d_i)$, $d_i \in D_i$, and $c(d_{ij}, d_{f_i(v_j)j})$ for each $v_j \in p_i$ and $d_i \in D_i$, with $1 \leq i \leq n$ and $1 \leq j \leq r$.*

Definition 2 *A solution of an instance of the inter-phase data layout problem is a set* $\{d_1, d_2, ...d_n\}$ *of data layout schemes* $d_i \in D_i, 1 \leq i \leq n$, *such that*

$$\sum_{i=1}^{n} c(P_i, d_i) \quad + \quad \sum_{i=1}^{n} \sum_{v_j \in p_i} c(d_{ij}, d_{f_i(v_j)j})$$

is minimized, where $c(d_{ij}, d_{ij})$ *is 0. Note that this implies not associating any cost with an initial data layout.*

Definition 2 results in an optimization problem. Section 3 contains the proof that the corresponding decision problem DYN-REMAP(k) is NP-complete.

Definition 3 *DYN-REMAP(k) represents a decision problem defined as follows:*

DYN-REMAP(k) := set of all instances of the inter-phase data layout problem such that there exists a set of data layouts $\{d_1, d_2, ...d_n\}$, $d_i \in D_i, 1 \leq i \leq n$, *with a cost less or equal to k, where k is a non-negative integer.*

3 NP-completeness Proof

Definition 4 *An instance of the 3 Conjunctive Normal Form Satisfiability Problem consists of a boolean expression B in conjunctive normal form,*

$$B = \bigwedge_{i=1}^{t} F_i,$$

where $F_i = l_i^1 \vee l_i^2 \vee l_i^3$, $1 \leq i \leq t$, *and each literal is a variable or its negation in the set of variables* $V = \{v_1, ...v_r\}$.
The decision problem 3SAT is represented as follows:

3SAT := set of all instances of the 3 Conjunctive Normal Form Satisfiability Problem for which there exists a truth value assignment $w : V \rightarrow \{\texttt{true}, \texttt{false}\}$ *such that B evaluates to* `true` *under w.*

Theorem 1 *DYN-REMAP(k) is NP-complete.*

Proof: The proof consists of two parts. Lemma 1 shows that DYN-REMAP(k) is in NP. Lemma 2 states that 3SAT can be reduced to DYN-REMAP(k) in polynomial time. Since 3SAT is NP-complete, DYN-REMAP(k) has to be NP-complete.
□

Lemma 1 *DYN-REMAP(k) is in NP.*

Proof: Let $\{d_1, d_2, ...d_n\}$, $d_i \in D_i, 1 \leq i \leq n$, be a set of data layouts for an instance of the inter-phase data layout problem, one data layout for each phase in the program. The overall cost of this set can be computed in polynomial time as described in Definition 2. Therefore it can be verified in polynomial time whether a given set of data layouts has a cost smaller or equal to a given cost k. Hence, DYN-REMAP(k) is in NP.
□

Lemma 2 *3SAT can be reduced in polynomial time to DYN-REMAP(k).*

Proof: Part 1 of the proof defines the function g that maps an instance B of the 3SAT problem onto an instance $g(B)$ of the DYN-REMAP(0) problem. Part 2 contains the proof that $B \in$ 3SAT $\Leftrightarrow g(B) \in$ DYN-REMAP(0). Finally, Part 3 shows that g can be computed in polynomial time.

Part 1: Let B be an arbitrary instance of the 3 Conjunctive Normal Form Satisfiability Problem, $B = \bigwedge_{i=1}^{t}(\ l_i^1 \vee l_i^2 \vee l_i^3 \ )$. g maps the instance B to an instance of the inter-phase data layout problem as follows:

- $V = \{v_1, ...v_r\}$, i.e. the sets of variables are the same.
- Each clause F_i is represented by a distinct phase P_i, $1 \leq i \leq t$.
- $p_i = \{v_j \mid l_i^k \ is \ a \ literal \ of \ variable \ v_j, \ 1 \leq k \leq 3\}$, where $1 \leq i \leq t$. Note that $|p_i| \leq 3$.
- Each variable $v \in p_i$ has 2 possible data layouts, called `T` and `F`. D_i contains $2^{|p_i|}$ candidate layouts, one layout for each possible combination of the single variable layouts. In other words, each $d_i \in D_i$ represents a truth value assignment w_i for all variables in p_i:

$$w_i(v_j) = \begin{cases} \texttt{true} & \text{if } d_{ij} = \texttt{T} \\ \texttt{false} & \text{if } d_{ij} = \texttt{F} \end{cases}$$

- Assume $D_i = \{d_i^1, ... \ d_i^m\}$.

$$c(P_i, d_i^k) = \begin{cases} 0 & \text{if } F_i \text{ is true under the truth value assignment represented by } d_i^k \ , \\ 1 & \text{otherwise} \ , \end{cases}$$

  where $1 \leq i \leq t$ and $1 \leq k \leq m$.

- Assume $d_{ij}^k \in d_i^k \in D_i$ and $d_{i'j}^{k'} \in d_{i'}^{k'} \in D_{i'}$, where $i' = f_i(v_j)$.

$$c(d_{ij}^k, d_{i'j}^{k'}) = \begin{cases} 0 & \text{if both data layouts are identical} \\ 1 & \text{otherwise} \end{cases}$$

  In other words, $c(d_{ij}^k, d_{i'j}^{k'}) = 0$ if and only if no remapping of v_j is required between the two data layouts.

Note that there are at most eight data layouts for each phase. An example of the application of g to an instance of 3SAT is given in Section 4.

Part 2a: Claim: $B \in$ 3SAT $\Rightarrow g(B) \in$ DYN-REMAP(0).

Proof: Let $w : V \rightarrow \{\texttt{true}, \texttt{false}\}$ be a truth value assignment that satisfies the problem instance B. There is exactly one data layout scheme in each phase P_i of $g(B)$ that represents

w restricted to the variables in p_i. Call this data layout scheme d'_i. For all i, $1 \leq i \leq t$, $c(P_i, d'_i) = 0$. Since w specifies a unique data layout for each single program variable $v_j \in V$, redistribution between the set of data layouts $\{d'_1, d'_2, \ldots d'_t\}$ does not occur. Therefore the set has an overall cost of 0. Hence $g(B) \in$ DYN-REMAP(0).

Part 2b: Claim: $g(B) \in$ DYN-REMAP(0) $\Rightarrow B \in$ 3SAT.

Proof: Let $\{d_1, d_2, \ldots d_t\}$ be a set of data layouts, one data layout for each phase P_i, with an overall cost of 0. Therefore no remapping can occur between the data layouts and each data layout d_i has to represent a truth value assignment that satisfies F_i. Hence, there exists a unique truth value assignment w that satisfies all $F_i, 1 \leq i \leq t$. The existence of such a truth value assignment means that B is in 3SAT.

Part 3: Claim: $g(B)$ can be computed in polynomial time.

Proof: The collection of functions f_i can be computed in $\mathcal{O}(t\ r)$, where t and r are the number of phases and program variables, respectively.

There are at most 8 data layouts per phase. Therefore there are at most $t*8$ data layout costs of the form $c(P_i, d_i^k)$ for the entire program. For each data layout d_i^k of phase P_i, at most $3*8$ remapping costs for individual variables $v_j \in p_i$ of the form $c(d_{ij}^k, d_{i'j}^{k'})$ have to be computed, resulting in at most $3*8^2$ remapping costs for each phase and at most $t*3*8^2$ for the entire program. Hence, g can be computed in polynomial time.
□

4 Example Reduction

The function g maps the instance $B = (v_1 \vee \neg v_2 \vee v_3) \wedge (\neg v_1 \vee v_2 \vee v_4) \wedge (v_1 \vee v_3 \vee \neg v_4)$ of the 3SAT problem into an instance of the decision problem DYN-REMAP(0) as follows:

- $V = \{v_1, v_2, v_3, v_4\}$
- There are three phases, P_1, P_2,and P_3. The ordering of the phases is given by their indices.
- $p_1 = \{v_1, v_2, v_3\}$, $p_2 = \{v_1, v_2, v_4\}$, and $p_3 = \{v_1, v_3, v_4\}$.
- $D_1 = \{ \{(v_1,F),(v_2,F),(v_3,F)\}, \{(v_1,F),(v_2,F),(v_3,T)\}, \{(v_1,F),(v_2,T),(v_3,F)\}, \{(v_1,F),(v_2,T),(v_3,T)\}, \{(v_1,T),(v_2,F),(v_3,F)\}, \{(v_1,T),(v_2,F),(v_3,T)\}, \{(v_1,T),(v_2,T),(v_3,F)\}, \{(v_1,T),(v_2,T),(v_3,T)\} \},$

  $D_2 = \{ \{(v_1,F),(v_2,F),(v_4,F)\}, \{(v_1,F),(v_2,F),(v_4,T)\}, \{(v_1,F),(v_2,T),(v_4,F)\}, \{(v_1,F),(v_2,T),(v_4,T)\}, \{(v_1,T),(v_2,F),(v_4,F)\}, \{(v_1,T),(v_2,F),(v_4,T)\}, \{(v_1,T),(v_2,T),(v_4,F)\}, \{(v_1,T),(v_2,T),(v_4,T)\} \},$ and

  $D_3 = \{ \{(v_1,F),(v_3,F),(v_4,F)\}, \{(v_1,F),(v_3,F),(v_4,T)\}, \{(v_1,F),(v_3,T),(v_4,F)\}, \{(v_1,F),(v_3,T),(v_4,T)\}, \{(v_1,T),(v_3,F),(v_4,F)\}, \{(v_1,T),(v_3,F),(v_4,T)\}, \{(v_1,T),(v_3,T),(v_4,F)\}, \{(v_1,T),(v_3,T),(v_4,T)\} \}.$

- The costs for the data layouts for the phases and the costs for remapping of individual variables is shown in Figure 1. Individual remapping costs are only shown for $d_3^5 \in D_3$, $d_3^5 = \{ (v_1, T), (v_3, F), (v_4, F) \}$. Each edge in the graph represents a cost function value $c(d_{3j}^5, d_{f_3(v_j)j}^k)$, $j \in \{1, 3, 4\}$.

Figure 2 shows a solution s to the example inter-phase data layout problem, $s = \{d_1^7, d_2^7, d_3^5\} \in$ DYN-REMAP(0). Note that all costs are 0. The corresponding truth value assignment is $\{(v_1, \texttt{true}), (v_2, \texttt{true}), (v_3, \texttt{false}), (v_4, \texttt{false})\}$. This truth value assignment satisfies B.

Acknowledgements

I would like to thank Ken Kennedy, John Mellor-Crummey, and Reinhard von Hanxleden for their helpful comments and suggestions.

References

[AL93] J. Anderson and M. Lam. Global optimizations for parallelism and locality on scalable parallel machines. In *Proceedings of the SIGPLAN '93 Conference on Program Language Design and Implementation*, Albuquerque, NM, June 1993.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[GS91] J.R. Gilbert and R. Schreiber. Optimal expression evaluation for data parallel architectures. *Journal of Parallel and Distributed Computing*, 13(1):58–64, September 1991.

[KK93] K. Kennedy and U. Kremer. Initial framework for automatic data layout in Fortran D: A short update on a case study. Technical Report CRPC-TR93-324-S, Center for Research on Parallel Computation, Rice University, July 1993.

[KMCKC93] U. Kremer, J. Mellor-Crummey, K. Kennedy, and A. Carle. Automatic data layout for distributed-memory machines in the D programming environment. In Christoph W. Kessler, editor, *Automatic Parallelization — New Approaches to Code Generation, Data Distribution, and Performance Prediction*, pages 136–152. Vieweg Advanced Studies in Computer Science, Verlag Vieweg, Wiesbaden, Germany, 1993. Also available as technical report CRPC-TR93-298-S, Rice University.

[LC90] J. Li and M. Chen. Index domain alignment: Minimizing cost of cross-referencing between distributed arrays. In *Frontiers90: The 3rd Symposium on the Frontiers of Massively Parallel Computation*, College Park, MD, October 1990.

[Mac87] M. E. Mace. *Memory Storage Patterns in Parallel Processing.* Kluwer Academic Publishers, Boston, MA, 1987.

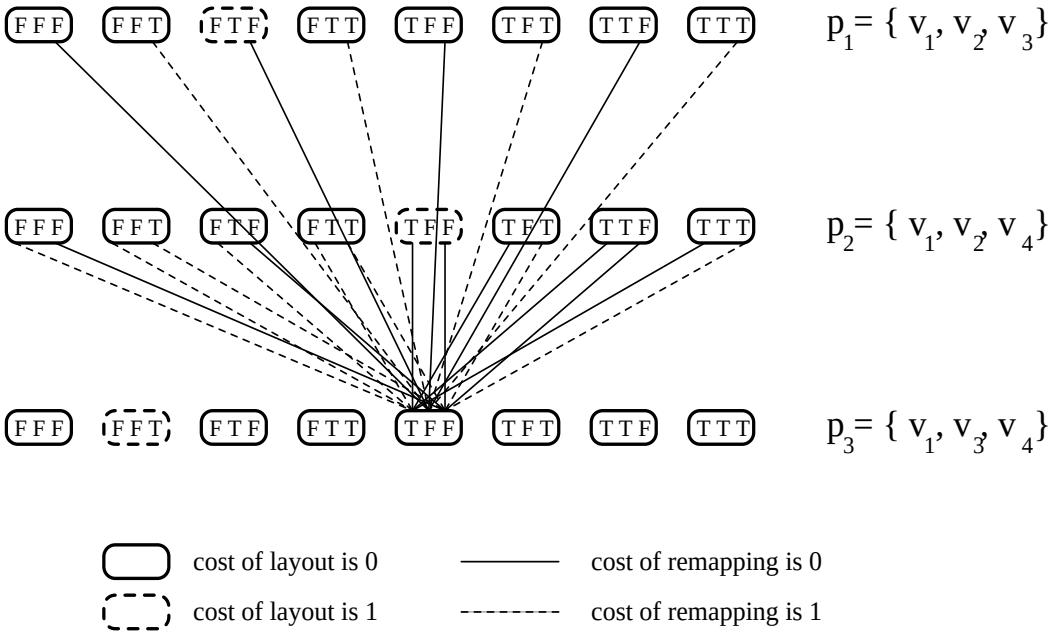

Figure 1: Sample costs for $g(B)$, $B = (v_1 \vee \neg v_2 \vee v_3) \ \wedge \ (\neg v_1 \vee v_2 \vee v_4) \ \wedge \ (v_1 \vee v_3 \vee \neg v_4)$

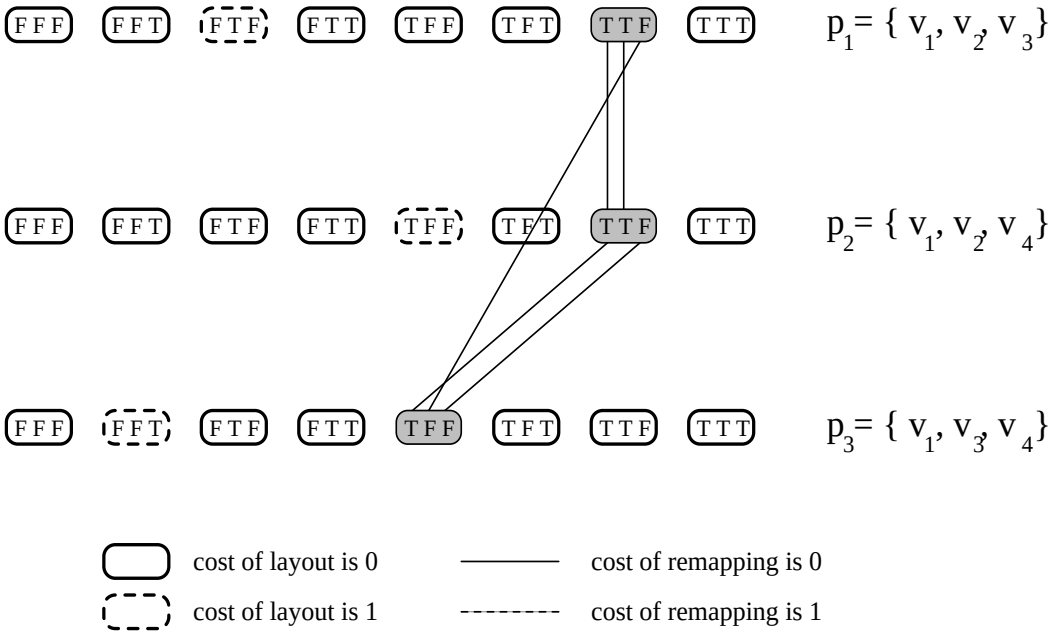

Figure 2: A solution for $g(B)$, $B = (v_1 \vee \neg v_2 \vee v_3) \ \wedge \ (\neg v_1 \vee v_2 \vee v_4) \ \wedge \ (v_1 \vee v_3 \vee \neg v_4)$