Włodzimierz BIELECKI, Krzysztof KRASKA, Maciej POLIWODA

West Pomeranian University of Technology

Dependence analysis and extraction of coarse-grained parallelism for parameterized perfectly-nested loops

Abstract. *This paper expands an approach presented in [1] to extraction of coarse-grained parallelism available in parameterized uniform and quasi-uniform perfectly-nested loops. It introduces a dependence analysis that is characterized by a polynomial time complexity and enables computing dependence distance vectors when the Petit dependence analyser fails to produce dependences. It permits to examine the performance of the approach for all parameterized perfectly-nested loops from the NAS Parallel Benchmark Suite.*

Streszczenie. *W artykule przedstawiono rozwinięcie zaprezentowanego we wcześniejszej pracy [1] podejścia do ekstrakcji gruboziarnistej równoległości w jednorodnych oraz quasi-jednorodnych pętlach programowych idealnie zagnieżdżonych. Rozwinięcie uprzednich wyników zostało osiągnięte poprzez wprowadzenie analizy zależności o wielomianowej złożoności obliczeniowej jednocześnie umożliwiającej obliczenie wektorów zależności w tych przypadkach, w których uprzednio wykorzystany analizator zależności Petit sygnalizował brak możliwości analizy źródła. Stworzyło to ostatecznie warunki do oceny skuteczności działania proponowanego podejścia dla wszystkich sparametryzowanych pętli idealnie zagnieżdżonych zawartych w zestawie testowym NAS Parallel Benchmark Suite (**Analiza zależności oraz ekstrakcja gruboziarnistej równoległości sparametryzowanych pętli idealnie zagnieżdżonych**).*

Keywords: parallelizing compilers, dependence analysis, quasi-uniform dependences, loop transformation.
Słowa kluczowe: kompilatory zrównoleglające, analiza zależności, zależności quasi-jednorodne, transformacja pętli.

Introduction

In paper [1] an approach is presented that enables the extraction of coarse-grained parallelism available in parameterized perfectly-nested static-control loops, where loop bounds as well as array subscripts are symbolic parameters. Most steps of the approach are characterized by a polynomial time complexity while other contemporary outstanding methods for extracting coarse-grained parallelism, such as [2, 3, 4], may require much time (even hundred of hours) and memory for calculations. This is why there exists a strong need in developing algorithms characterized by reduced time and memory complexities. Carried out experiments by the authors for the NAS Parallel Benchmark Suite [5] demonstrate that the proposed approach is efficient and very fast. In [1] the authors use the Petit dependence analysis tool [6], which fails to produce dependences for 28 loops. Moreover, Petit's calculations are based on the Presburger arithmetic which in general is not characterized by polynomial time complexity.

The main purpose of this paper is to expand the approach presented in [1] by introducing a dependence analysis that is characterized by a polynomial time complexity and to examine the performance of the approach for all parameterized perfectly-nested loops from the NAS Parallel Benchmark Suite.

Background

In this section, we briefly introduce necessary preliminaries which are used throughout this paper.

The following concepts of linear algebra are used in the approach presented in this paper: a polyhedron, lattice, the Hermite Normal Form of the matrix and its uniqueness, Hermite decomposition, affine lattice canonical form. Details can be found in papers [7, 8].

Definition 1 (Congruence relation, modulus matrix). Let y and z be two d-dimensional integral vectors and D be some integral $d \times d$ matrix of full row rank. We say that y is congruent to z modulo the column image of D, written:

$$y \equiv z \mod D,$$

if and only if the difference $z - y$ is equal $D \times x$ for some d-dimensional integral vector $x \in Z^d$. Matrix D is called the modulus matrix [9].

Definition 2 (Equivalence relation). Matrix D yields an equivalence relation, denoted by $\sim_D$, which is defined by $y \sim_D z$ if and only if $y \equiv z \bmod D$ [9].

Definition 3 (Equivalence class). An equivalence class in a set is the subset of all elements which are in equivalence relation $\sim_D$. The number of equivalence classes of $\sim_D$, is denoted by *vol(D)*, the volume of D, which is the absolute value of the determinant of D. If the determinant of D is zero, then D does not have full row rank and thus $\sim_D$ has the infinite number of equivalence classes. An equivalence class of $\sim_D$ is also called a lattice [9].

Definition 4 (Representatives). The set of all integral vectors in the parallelepiped *R(D)* defined by the columns of D:

$$R(D) = \left\{x \in \mathrm{Z} \mid x = D \cdot \alpha, \alpha \in R^d, 0 \le \alpha \le 1\right\},$$

defines a set of representatives for the equivalence classes of $\sim_D$ [9].

In this paper, we deal with the following definitions concerned program loops: iteration space (IS), loop domain (index set), parameterized loops, perfectly-nested loops, dependence, dependence distance set, dependence distance vector, uniform dependence, non-uniform dependence, whose explanations are given in papers [3, 10].

Definition 5 (Uniform loop, quasi-uniform loop). We say that a parameterized loop is uniform if it induces dependences represented with the finite number of uniform dependence distance vectors [3]. A parameterized loop is quasi-uniform if all its dependence distance vectors can be represented by a linear combination of the finite number of linearly independent vectors with constant coordinates.

Let us consider the parameterized dependence distance vector *(N,2)*. It can be represented as the linear combination of the two linearly independent vectors *(0,2)* and *(1,0)* as follows *(0,2)* + a $\times$ *(1,0)*, where $a \in Z$.

Definition 6 (Dependence Sources Polyhedron). Given a dependence distance vector $d_{S,T}$, a dependence sources polyhedron *DSP*($d_{S,T}$) is the set of all the values of iteration vector $\overrightarrow{i_S}$ such that there exist dependences between $S\left(\overrightarrow{i_S}\right)$ and $T\left(\overrightarrow{i_S} + d_{S,T}\right)$.

Definition 7 (Polyhedral Reduced Dependence Graph). A Polyhedral Reduced Dependence Graph (PRDG) is the graph where a vertex stands for every statement *S* and an edge connects statements *S* and *T* whose instances are dependent. The number of edges between vertices *S* and *T* is equal to the number of vectors $d_{S,T} \in D_{S,T}$. Every edge is labelled with a dependence distance vector $d_{S,T}$ and a dependence sources polyhedron $P(d_{S,T})$.

In the next section of the paper, we analyse the time complexity of the proposed approach in a machine-independent way of assessing the performance of algorithms. For this purpose, the RAM (Random Access Machine) model of computation is used. Under the RAM model, we measure run-time by counting up an upper bound, O, on the number of steps an algorithm takes on a given problem instance. Details on the model and the time complexity analysis can be found in paper [11].

Approach to calculating dependence distance vectors and extracting parallelism

In this section, we expand the approach presented in [1] to extraction of equivalence classes for both uniform and quasi-uniform loops on the step of calculating dependence distance vectors which is characterized by a polynomial time complexity.

Calculating dependence distance vectors

To find dependence distance vectors, a system of equations should be built for each pair of the same named variables $ID(A_1\overline{I} + B_1)$, $ID(A_2\overline{I} + B_2)$ that are located in the loop body on both hand sides of assignment statements, the right and the left, or on the left-hand sides only, where A_1, A_2 are matrices of dimensions $m \times n$, B_1, B_2 are *m*-dimensional vectors. This system can be written as follows:

$$\begin{cases} A_1\overline{I} - A_2\overline{J} = \overline{B}_2 - \overline{B}_1 \\ \overline{D} = \overline{J} - \overline{I} \\ \overline{D} \succ \overline{0} \end{cases} . \tag{1}$$

For a pair of dependent iterations, the source is the iteration that is lexicographically less.

In system (1), vector $\overline{I}$ describes all the iterations that form the sources of pairs of dependent iterations, while vector $\overline{J} = \overline{I} + \overline{D}$ describes destinations of those, $\overline{D}$ is the dependence vector. To obtain correct results, all dependent iterations have to be executed in lexicographical order [1].

To determine vector $\overline{D}$, we have to solve system (1). From the second equation, we have:

$$\overline{J} = \overline{I} + \overline{D} . \tag{2}$$

Substituting (2) into the first equation of (1), we obtain:

$$A_2\overline{D} + (A_2 - A_1)\overline{I} = \overline{B}_1 - \overline{B}_2 . \tag{3}$$

A solution to equation (3) is a dependence distance vector *D* with integer coordinates. When there does not exist any solution to equation (3), the loop does not expose any dependences. Equation (3) can be solved by means of the Gaussian elimination algorithm whose time complexity is discussed later.

Consider the following loop:

```
for(i = 1; i <= 4; i++)
  for(j = 1; j <= 3; j++)
    a[i][j] = a[i+1][j]+a[i][j+1];
```

For this loop, there exist two dependence vectors: $\overline{D}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $D_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

Extracting equivalence classes for both uniform and quasi-uniform loops

Input : A parameterized perfectly--nested uniform or quasi-uniform loop
Output : The parallelized loops scanning independent code fragments

Method:

1) *Calculate dependence distance vectors*. Find dependencies in parameterized perfectly-nested uniform or quasi-uniform loop. Work out dependence distance vectors using the approach presented above.
2) *Replace all parameterized dependence distance vectors.* Replace each parameterized dependence distance vector with a linear combination of vectors with constant coordinates. For this purpose apply the algorithm presented in [12].
3) *Form a dependence distance set.* Form matrix *D*, $D \in Z^{n \times m}$, whose *m* columns are all non-parameterized dependence distance vectors $d_{S,T}$, corresponding to the edges of *PRDG*. Associate row *k* of $D^{n \times m}$ with a loop index i_k, *k=1,2,…,n* where *n* is the number of loop indices (i.e. surrounding loops).
4) *Form the basis of the lattice from the dependence distance set.* Transform matrix *D* into two sub-matrices *D'*, $D' \in Z^{l \times m}$ and *D''*, $D'' \in Z^{(n-l) \times m}$, such that *l* rows of *D'*, $1 \le l \le n$, are linearly independent and *(n - l)* rows of *D''* are linearly dependent. When interchanging two rows, interchange also the loop indices associated with these rows.
5) *Find lattice canonical form*. Transform sub-matrix *D'* to the Hermite Normal Form:

$$D' = H \times U = \begin{bmatrix} B & 0 \end{bmatrix} \times U, B \in Z^{l \times l} ,$$

preserving loop indices associated with the rows of *D'*. Note that the lattice canonical form represents the equivalence relation.
6) *Find representatives for equivalence classes.* Using *B*, calculate a set of representatives (one for each equivalence class) depending on the following cases:
   a) *l=n*: define a set *R(B)* of representatives for equivalence classes as the set of all integral vectors in the parallelepiped defined by the columns of *B*,

$$R(B) = \left\{ h \in Z \mid h = B \cdot \alpha, \alpha \in R^d, 0 \le \alpha \le 1 \right\},$$

   b) *l<n*: find the first *l* coordinates of representatives for equivalence classes as follows:

$$R(B^l) = \left\{ h^l \in Z^l \mid h^l = B^l \cdot \alpha, \alpha \in R^l, 0 \le \alpha \le 1 \right\},$$

   and enlarge matrix B^l to matrix *B* by inserting the last *n-l* zero rows.

7) *Find equivalence classes.* Using representatives *h*, $h \in R(B)$, form the following polyhedra that specifies equivalence classes:

$$\mathrm{P} = \left\{ y \mid y = h + B \times z \text{ s.t. } h \in R(B) \wedge z \in Z^n \wedge y \in IS \right\}.$$

Each equivalence class represents an independent subset of statement instances which are represented by an equivalence relation.

8) *Generate parallel loops.* Apply any well-known code generation algorithm to generate parallel outer loops scanning a set of representatives *R(B)* and sequential inner loops enumerating in the lexicographical order the elements of set *P* for each equivalence class represented by *h*.

The degree of parallelism is characterized by the number of equivalence classes and can investigated by a simple inspection of matrix *B*. Look at the paper [1] for details.

Time complexity

Most steps of the proposed approach can be accomplished in polynomial time.

1) The task of calculating dependence distance vectors requires solving a system of linear equations and it can be done in polynomial time by the Gaussian elimination. According to [13], this computation can be done in $O(ldm)$ arithmetic operations.
2) The task of identifying a set of linearly independent rows of a matrix *D*, $D \in Z^{n \times m}$, with constant coordinates and dependent ones can be done in polynomial time by the Gaussian elimination. This computation can be done in $O(ldm)$ arithmetic operations, as it was previously mentioned.
3) The task of transforming a matrix *D'*, $D' \in Z^{l \times m}$ to its Hermite Normal Form matrix *B*, $B \in Z^{l \times l}$ can be accomplished, depending on the algorithm used, even in $O\left(l^{\theta-1} m \log(2m/l) B(l \log(l \|D'\|))\right)$ operations [13], where $\theta < 2.376$.
4) A set *R(B)* of representatives for equivalence classes is defined as a set of all integral vectors in the parallelepiped defined by the columns of *B*. According to [9] such a set of representatives can be found by enumerating the equivalence classes with nonnegative integral vectors in the lexicographical order ($y <_{lex} z$ if there is some i, $1 \le i \le l$, such that $y_i < z_i$ and for $j=1,\ldots,l-1$ and $y_j=z_j$). Enumerating l diagonal coefficients of *B* requires $O(l)$ operations and enumerating n-l rows of *D''* requires $O(n \times (n-l))$ operations.

Experiments

We have put the presented dependence analysis module into the C++ implementation of the approach to the extraction of equivalence classes. Additionally, we have used the well-known tool CLooG v0.14.1 [14] for code generation to achieve output source codes. The C++ codes have been compiled using the gcc v4.5.0 compiler.

In order to evaluate the performance of the presented approach, we have examined all the perfectly-nested quasi-uniform loops (contained parameterized dependence distance vectors) provided by the well-known NAS Parallel Benchmark (NPB) Suite. In NPB, we have found 185 perfectly nested loops distributed in terms of dependence types as shown in Table 1.

Table 1. The quantitative distribution of perfectly-nested loops in terms of dependence types for the NAS Parallel Benchmark Suite

	The results of dependence analysis		Number of loops
1)	No dependences	:	123
2)	Uniform dependences, including:	:	14
	• non loop-carried dependences	:	9
	• loop-carried dependences	:	5
3)	Affine dependence distance vectors	:	1
4)	Parameterized dependence distance vector	:	**47**
	The total number of perfectly nested loops	:	185

Table 2. Effectiveness and time analyses of the proposed approach for quasi-uniform loops from the NAS Parallel Benchmark Suite

#	Source loop	Degree of parallelism	Number of parameterized vectors	Dependence analyzer	Petit	Time taken by the main steps of the approach [μs]					
						- 2 -	- 3 -	- 4 -	- 5 -	- 6 -	- 7 -
1)	BT_error.f2p_5.t	1	29	308	N/A	3364	2	277	53	-	-
2)	BT_rhs.f2p_1.t	1	39	849	N/A	3919	2	292	67	-	-
3)	BT_rhs.f2p_5.t	1	111	3608	N/A	6476	5	879	223	-	-
4)	CG_cg.f2p_4.t	1	8	245	N/A	433	1	51	11	-	-
5)	FT_auxfnct.f2p_1.t	1	1	126	1040	56	1	6	5	-	-
6)	LU_HP_jacld.f2p_1.t	1	1761	95279	N/A	82021	27	12532	2325	-	-
7)	LU_HP_jacu.f2p_1.t	1	1761	92593	N/A	79148	27	11141	2374	-	-
8)	LU_HP_l2norm.f2p_2.t	1	9	248	N/A	999	1	49	13	-	-
9)	LU_HP_pintgr.f2p_11.t	1	6	426	N/A	693	1	31	9	-	-
10)	LU_HP_pintgr.f2p_2.t	1	88	486	N/A	4890	5	879	153	-	-
11)	LU_HP_pintgr.f2p_3.t	1	6	426	N/A	623	2	41	11	-	-
12)	LU_HP_pintgr.f2p_7.t	1	6	385	N/A	591	1	39	8	-	-
13)	LU_jacld.f2p_1.t	1	2080	92323	N/A	87734	25	21340	3729	-	-
14)	LU_jacu.f2p_1.t	1	2080	92689	N/A	89439	29	19880	3920	-	-
15)	LU_l2norm.f2p_2.t	5	9	271	N/A	896	6	54	17	1	192
16)	LU_pintgr.f2p_11.t	1	6	285	N/A	702	1	30	11	-	-
17)	LU_pintgr.f2p_2.t	1	88	486	N/A	5937	5	930	186	-	-
18)	LU_pintgr.f2p_3.t	1	6	285	N/A	673	1	39	11	-	-
19)	LU_pintgr.f2p_7.t	1	6	285	N/A	629	1	40	11	-	-
20)	SP_error.f2p_5.t	1	29	308	N/A	3092	2	298	61	-	-
21)	SP_ninvr.f2p_1.t	1	87	1563	N/A	6088	5	721	161	-	-
22)	SP_pinvr.f2p_1.t	1	87	1643	N/A	6551	4	798	156	-	-
23)	SP_rhs.f2p_1.t	1	54	1072	N/A	4608	4	492	111	-	-
24)	SP_rhs.f2p_5.t	1	111	3608	N/A	5760	2	1013	236	-	-
25)	SP_txinvr.f2p_1.t	1	234	2671	N/A	8573	6	1997	411	-	-
26)	SP_tzetar.f2p_1.t	1	249	2749	N/A	7980	9	2394	469	-	-

27)	UA_adapt.f2p_2.t	*N1×N3×N4*	24	2691	22981	2416	2	267	51	1	221
28)	UA_diffuse.f2p_1.t	1	12	447	N/A	1167	2	69	21	-	-
29)	UA_diffuse.f2p_2.t	1	3	215	17037	148	1	18	6	-	-
30)	UA_diffuse.f2p_3.t	*N1×N3×N4*	3	257	10960	346	1	23	6	2	204
31)	UA_diffuse.f2p_4.t	*N1×N3×N4*	3	251	3769	312	2	21	5	1	225
32)	UA_diffuse.f2p_5.t	*N2×N3×N4*	3	262	3523	348	1	21	5	1	238
33)	UA_precond.f2p_3.t	*N1*	3	258	2448	237	1	15	6	1	202
34)	UA_precond.f2p_5.t	1	32	189	N/A	3795	2	369	68	-	-
35)	UA_setup.f2p_16.t	*N1×N2*	3	229	1218	321	1	20	6	1	221
36)	UA_transfer.f2p_1.t	1	3	176	1837	197	1	15	6	-	-
37)	UA_transfer.f2p_2.t	1	3	161	1274	170	1	13	6	-	-
38)	UA_transfer.f2p_3.t	1	3	169	1281	168	1	13	6	-	-
39)	UA_transfer.f2p_5.t	1	3	160	1248	170	1	14	5	-	-
40)	UA_transfer.f2p_6.t	1	3	168	1275	167	1	13	6	-	-
41)	UA_transfer.f2p_7.t	*N1*	3	192	2413	269	1	16	7	1	202
42)	UA_transfer.f2p_8.t	1	3	161	1249	169	1	16	8	-	-
43)	UA_transfer.f2p_9.t	*N1*	3	195	9182	270	1	18	5	1	199
44)	UA_transfer.f2p_10.t	1	3	160	1259	173	1	14	6	-	-
45)	UA_transfer.f2p_13.t	*N1*	3	195	2468	262	1	15	5	1	203
46)	UA_transfer.f2p_15.t	*N1*	3	233	2763	257	1	15	6	1	202
47)	UA_transfer.f2p_18.t	*N1*	3	236	2768	256	1	15	6	1	196

Quasi-uniform loops (contained parameterized distance vectors) have been parallelized using the Intel PentiumM 1.5GHz machine with the Linux openSUSE v11.1 32-bit operating system. The results of the experiments are collected in Table 2, where time is presented in microseconds and N/A stands for not-available.

Using the presented dependence analysis, we have managed to find dependence vectors for all parameterized perfectly-nested loops and decrease the time of a parallel compilation. Under our experiments, we have found that a dependence analysis as well as the replacement of parameterized dependence distance vectors by a linear combination of constant vectors have taken most of the time. The other steps of the algorithm have performed several times faster. The whole time required for a dependence analysis and extracting equivalence classes is counted up in milliseconds, what is similar to the previously obtained results.

Conclusions

In this paper, we have expanded the approach presented in [1] by introducing a dependence analysis characterized by a polynomial time complexity. It has permitted to examine the performance of the approach for all parameterized perfectly-nested loops from the NAS Parallel Benchmark Suite. The experiments confirm that the presented approach is very fast and the use of efficient dependence analysis methods can improve its effectiveness and the performance of parallel compilation.

The comparison of the performance of the presented approach with other well-known techniques is under our current research. In our next work, we plan to extend the approach to imperfectly nested loops and investigate its effectiveness and time complexity.

Wydanie publikacji zrealizowano przy udziale środków finansowych otrzymanych z budżetu Województwa Zachodniopomorskiego.

REFERENCES

[1] Bielecki, W., Kraska, K., Extracting coarse-grained parallelism for affine perfectly nested quasi-uniform loops. *9th International Conference on Parallel Processing and Applied Mathematics, proceedings: R. Wyrzykowski et al. (Eds.): PPAM 2011, Part I, LNCS 7203* (2012), 307-316

[2] Lim, A.W., Lam M.S., Maximizing Parallelism and Minimizing Synchronization with Affine Partitions, *Parallel Computing*, (1998), Vol 24, Issue 3-4, 445-475

[3] Griebl, M., Automatic Parallelization of Loop Programs for Distributed Memory Achitectures. Habilitation, Fakultät für Mathematik und Informatik Universität Passau (2004)

[4] Beletska, A., Bielecki, W., Pietro, P.S., Extracting Coarse-Grained Parallelism in Program Loops with the Slicing Framework, *In ISPDC* (2007), 203-210

[5] NASA Advanced Supercomputing Division, http://www.nas.nasa.gov

[6] Kelly, W., Maslov, V., Pugh, W., Rosser, E., Shpeisman, T., Wonnacott, D., The Omega Library Interface Guide, *Technical Report CS-TR-3445*, Dept. of Computer Science, University of Maryland College Park (1995)

[7] Schrijver, A., Theory of Linear and Integer Programming, *Series in Discrete Mathematics* (1999)

[8] Nookala, P.K.V.V, Risset, T., A Library for Z-Polyhedral Operations, *Publication interne* n.1330, Institut de Recherche en Informatique et Systèmes Aléatoires (2000)

[9] Hofting, F., Wanke, E., Polynomial-Time Analysis of Toroidal Periodic Graphs, *Journal of Algorithms* 34 (2000), 14-39

[10] Bondhugula, U.K.R.,Effective Automatic Parallelization and Locality Optimization Using the Polyhedral Model. Dissertation, The Ohio State University (2010)

[11] Skiena, S., The Algorithm Design Manual. 2nd Edition, *Springer* (2008)

[12] Quinton, P., Rajopadhye, S., Risset, T., On Manipulating Z-Polyhedra. *Publication interne* n.1016, Institut de Recherche en Informatique et Systèmes Aléatoires (1996)

[13] Cohen, E., Megiddo, N., Recognizing Properties of Periodic Graphs. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 4 (1991), 135-146

[14] Bastoul, C., Code Generation in the Polyhedral Model Is Easier Than You Think, *PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques* (2004), 7-16

***Authors**: prof. dr hab. inż. Włodzimierz Bielecki, Zachodniopomorski Uniwersytet Technologiczny, Wydział Informatyki, ul. Żołnierska 49, 71-210 Szczecin, E-mail: wbielecki@wi.zut.edu.pl; dr inż. Krzysztof Kraska, Zachodniopomorski Uniwersytet Technologiczny, Wydział Informatyki, ul. Żołnierska 49, 71-210 Szczecin, E-mail: kkraska@wi.zut.edu.pl; dr inż. Maciej Poliwoda, Zachodniopomorski Uniwersytet Technologiczny, Wydział Informatyki, ul. Żołnierska 49, 71-210 Szczecin, E-mail: mpoliwoda@wi.zut.edu.pl.*