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 zagnieźdzonych. 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 źródeł. Stworzono to ostatecznie warunki do oceny skuteczności działania proponowanego podejścia dla wszystkich sparamezyzowanych pętli idealnie zagnieźdzonych zawartych w zestawie testowym NAS Parallel Benchmark Suite (Analiza zależności oraz ekstrakcja gruboziarnistej równoległości sparamezyzowanych pętli idealnie zagnieźdzonych).

Keywords: parallelizing compilers, dependence analysis, quasi-uniform dependences, loop transformation.

Słowa kluczowe: kompilatory równoległe, 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:

\[ y = z \mod D, \]

if and only if the difference \(z - y\) is equal \(D \times x\) for some \(d\)-dimensional integral vector \(x \in \mathbb{Z}^d\). 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 \(y = z \mod D\) [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 \(\text{vol}(D)\), 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 parallellepiped \(R(D)\) defined by the columns of \(D\):

\[ R(D) = \{x \in \mathbb{Z} | x = D \cdot \alpha, \alpha \in \mathbb{R}^d, 0 \leq \alpha \leq 1\}, \]

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 \((0,2) + a \cdot (1,0)\), where \(a \in \mathbb{Z}\).

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

PL "Stworzy zagnieżdżenia w jednorodnych oraz quasi-jednorodnych pętlach programowych idealnie zagnieźdzonych. 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 źródeł. Stworzono to ostatecznie warunki do oceny skuteczności działania proponowanego podejścia dla wszystkich sparamezyzowanych pętli idealnie zagnieźdzonych zawartych w zestawie testowym NAS Parallel Benchmark Suite (Analiza zależności oraz ekstrakcja gruboziarnistej równoległości sparamezyzowanych pętli idealnie zagnieźdzonych)."
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 $d_{ST} \in D_{ST}$. Every edge is labelled with a dependence distance vector $d_{ST}$ and a dependence sources polyhedron $P(d_{ST})$.

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 $ID(A_1 \bar{T} + B_1)$, $ID(A_2 \bar{T} + B_2)$ that are located in the loop body on both hands 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:

$$
\begin{align*}
\text{(1)} \quad & A_1 \bar{T} - A_2 \bar{T} = B_2 - B_1 \\
& \bar{D} = \bar{T} - \bar{T} \\
& \bar{D} > 0
\end{align*}
$$

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

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

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

$$
\text{(2)} \quad \bar{J} = \bar{T} + \bar{D}.
$$

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

$$
\text{(3)} \quad A_2 \bar{D} + (A_2 - A_1) \bar{T} = B_1 - B_2.
$$

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:

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

For this loop, there exist two dependence vectors:

$$
\bar{D}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \text{and} \quad \bar{D}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.
$$

Extracting equivalence classes for both uniform and quasi-uniform loops

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 $d_{ST}$, corresponding to the edges of PRDG. Associate row $k$ of $D^{n \times m}$ with a loop index $i_k$, $k=1,2,\ldots,n$ 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 $D', D'' \in Z^{l \times m}$, such that $l \leq n$, 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:

$$
D' = H \times U = [B \ 0] \times U, B \in Z^{l \times l},
$$

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:

- $l=n$: 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 $B$,

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

- $l<n$: find the first $l$ coordinates of representatives for equivalence classes as follows:

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

and enlarge matrix $B'$ 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:

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

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 be 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 $O(d\text{mand})$ arithmetic operations.

2) The task of identifying a set of linearly independent rows of a matrix $D$, $D \in \mathbb{Z}^{n\times m}$, with constant coordinates and dependent ones can be done in polynomial time by the Gaussian elimination. This computation can be done in $O(d\text{mand})$ arithmetic operations, as it was previously mentioned.

3) The task of transforming a matrix $D'$, $D' \in \mathbb{Z}^{n\times m}$ to its Hermite Normal Form matrix $B$, $B \in \mathbb{Z}^{n\times m}$ can be accomplished, depending on the algorithm used, even in $O\left(\theta^{-1}\log(2m/n)B(\log(lD'))\right)$ 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 $(y \leq z$ if there is some $i$, $1 \leq i \leq l$, 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 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>
<thead>
<tr>
<th>#</th>
<th>Source loop</th>
<th>Degree of parallelism</th>
<th>Number of parameterized vectors</th>
<th>Dependence analyzer</th>
<th>Petl</th>
<th>Time taken by the main steps of the approach [μs]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BT_error.f2p.5.t</td>
<td>1</td>
<td>29</td>
<td>308</td>
<td>N/A</td>
<td>3364</td>
</tr>
<tr>
<td>2</td>
<td>BT_rhs.f2p.1.t</td>
<td>1</td>
<td>39</td>
<td>849</td>
<td>N/A</td>
<td>3919 292 67</td>
</tr>
<tr>
<td>3</td>
<td>BT_rhs.f2p.5.t</td>
<td>1</td>
<td>111</td>
<td>3608</td>
<td>N/A</td>
<td>6476 879 223</td>
</tr>
<tr>
<td>4</td>
<td>CG cg.f2p.4.t</td>
<td>1</td>
<td>8</td>
<td>249</td>
<td>N/A</td>
<td>433 51 11</td>
</tr>
<tr>
<td>5</td>
<td>FT_auxfnct.f2p.1.t</td>
<td>1</td>
<td>1</td>
<td>126</td>
<td>N/A</td>
<td>36 5 6</td>
</tr>
<tr>
<td>6</td>
<td>LU_rhs.f2p.1.t</td>
<td>1</td>
<td>1761</td>
<td>95279</td>
<td>N/A</td>
<td>82021 12532 2325</td>
</tr>
<tr>
<td>7</td>
<td>LU_rhs.f2p.5.t</td>
<td>1</td>
<td>1761</td>
<td>92593</td>
<td>N/A</td>
<td>79148 11141 2374</td>
</tr>
<tr>
<td>8</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>9</td>
<td>249</td>
<td>N/A</td>
<td>999 49 13</td>
</tr>
<tr>
<td>9</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>6</td>
<td>426</td>
<td>N/A</td>
<td>693 31 9</td>
</tr>
<tr>
<td>10</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>8</td>
<td>486</td>
<td>N/A</td>
<td>4890 879 153</td>
</tr>
<tr>
<td>11</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>6</td>
<td>426</td>
<td>N/A</td>
<td>623 41 11</td>
</tr>
<tr>
<td>12</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>6</td>
<td>385</td>
<td>N/A</td>
<td>391 5 8</td>
</tr>
<tr>
<td>13</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>2080</td>
<td>92323</td>
<td>N/A</td>
<td>87734 21340 3729</td>
</tr>
<tr>
<td>14</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>2080</td>
<td>92689</td>
<td>N/A</td>
<td>89439 29 19880 3920</td>
</tr>
<tr>
<td>15</td>
<td>LU_rhs.f2p.2.t</td>
<td>5</td>
<td>9</td>
<td>271</td>
<td>N/A</td>
<td>896 54 17 192</td>
</tr>
<tr>
<td>16</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>6</td>
<td>285</td>
<td>N/A</td>
<td>702 30 11</td>
</tr>
<tr>
<td>17</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>8</td>
<td>486</td>
<td>N/A</td>
<td>5903 900 186</td>
</tr>
<tr>
<td>18</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>6</td>
<td>285</td>
<td>N/A</td>
<td>673 39 11</td>
</tr>
<tr>
<td>19</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>6</td>
<td>285</td>
<td>N/A</td>
<td>629 40 11</td>
</tr>
<tr>
<td>20</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>29</td>
<td>308</td>
<td>N/A</td>
<td>3092 298 61</td>
</tr>
<tr>
<td>21</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>87</td>
<td>1563</td>
<td>N/A</td>
<td>6088 5 721 161</td>
</tr>
<tr>
<td>22</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>87</td>
<td>1643</td>
<td>N/A</td>
<td>6551 798 156</td>
</tr>
<tr>
<td>23</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>54</td>
<td>1072</td>
<td>N/A</td>
<td>4608 492 111</td>
</tr>
<tr>
<td>24</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>111</td>
<td>3608</td>
<td>N/A</td>
<td>5760 2013 236</td>
</tr>
<tr>
<td>25</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>234</td>
<td>2671</td>
<td>N/A</td>
<td>6573 6 1997 411</td>
</tr>
<tr>
<td>26</td>
<td>LU_rhs.f2p.2.t</td>
<td>1</td>
<td>249</td>
<td>2749</td>
<td>N/A</td>
<td>7980 9 2394 469</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Number of loops</th>
<th>-2 -</th>
<th>-3 -</th>
<th>-4 -</th>
<th>-5 -</th>
<th>-6 -</th>
<th>-7 -</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) No dependencies</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2) Uniform dependences, including:</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• non loop-carried dependences</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• loop-carried dependences</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3) Affine dependence distance vectors</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4) Parameterized dependence distance vector</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The total number of perfectly nested loops: 185
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 been 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.

**REFERENCES**


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