Elektrotehniški vestnik 76(1-2): 13–18, 2009
Electrotechnical Review, Ljubljana, Slovenija

Ugotavljanje podatkovne odvisnosti za procesorje z naborom ukazov SIMD

Patricio Bulić, Tomaž Dobravec

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Tržaška 25, 1000 Ljubljana, Slovenija

Povzetek. V tem članku predstavimo algoritem za ugotavljanje obstoja podatkovne odvisnosti pri vektorizaciji zank za CPE z naborom ukazov SIMD. Znano je, da lahko zaporedje ukazov, ki izvedejo enako operacijo nad sosednjimi operandi v pomnilniku, nadomestimo z enim samim ukazom SIMD, če med temi ukazi ni prave podatkovne odvisnosti, tj. odvisnosti tipa RAW. Vendar se izkaže, da lahko pravo podatkovno odvisnost ignoriramo, če je le razdalja med pomnilniškimi referencami, ki jih beremo, in pomnilniškimi referencami, v katere pišemo, enaka ali večja od dolžine registrov SIMD v CPE oz. od števila operandov, nad katerimi se naenkrat izvede operacija SIMD. Z ustrezno analizo linearnih izrazov, s katerimi naslavljamo pomnilniške lokacije, naš algoritem ugotavlja kolikšna je razdalja med pomnilniškimi referencami, ki nastopata v dveh potencialno podatkovno odvisnih ukazih. Na podlagi te razdalje predlagani postopek, ki temelji na preverjanju minimuma in maksimuma razdalje med odvisnima pomnilniškima referencama, daje zeleno luč za vektorizacijo zank med postopkom prevajanja. Predlagani postopek spada v skupino t.i. približnih algoritmov za ugotavljanje podatkovne odvisnosti, saj poskuša le dokazati, da odvisnosti ni. Izkaže se, da je predlagani postopek učinkovitejši kot najpogosteje uporabljen približni Banerjeejev postopek in ga zato lahko nadomesti v hierarhičnem sistemu za ugotavljanje podatkovne odvisnosti.

Ključne besede: podatkovna odvisnost, multimedijske razširitve ISA, nabor ukazov SIMD, vektorizacija zank

Testing data dependency for microprocessors with a short SIMD instruction set

Extended abstract. In this paper we present an algorithm for the data-dependency problem related to microprocessors with a multimedia extension (i.e., including short SIMD instructions). Actually, there is a number of data-dependency tests proposed in literature (Banerjee test [2], GCD test [11], Omega test [7], Power test [10], etc.) which are all based on solving the linear dependence system. These tests would prohibit any vectorization if dependence exists even though this dependence would not be violeted after the vectorization. As an extension to the Banerjee test, the presented method checks whether the vectorization affects any existing depenedence relation by checking the distance between two conflicting memory references. We assume a p-nested for loop. In short SIMD processing, we can process V_l data simultaneously and these data must be successively stored in the memory. All the data must be of the same bit length b and the SIMD registers must be $V_l \cdot b$ bits long. Thus, the problem of SIMD vectorization becomes a problem of whether we can unroll the innermost loop V_l times and substitute the simultaneous statements with a single SIMD statement that performs the same operation at a time, over V_l b-bits data in the $(V_l \cdot b)$-bits SIMD register.
In this paper, we prove that the distance between the memory references in the innermost loop, which is defined as $d(\mathbf{i}^{'}, \mathbf{i}^{''}) = i_p^{''} - i_p^{'}$, is

$$d(\mathbf{i}^{'}, \mathbf{i}^{''}) = -a_p^{''}\zeta(\mathbf{i}^{'}, \mathbf{i}^{''}) - |a_p^{'} - a_p^{''}|i_p^{'}.$$

SIMD vectorization can be performed iff the distance between the memory references in the innermost loop is greater than or equal to the number of data processed in the SIMD register (V_l), i.e.

$$\max_{i',i''\in I^p} -a_p^{''}\zeta(\mathbf{i}^{'}, \mathbf{i}^{''}) - |a_p^{'} - a_p^{''}|i_p^{'} \leq 0$$

or

$$\min_{i',i''\in I^p} -a_p^{''}\zeta(\mathbf{i}^{'}, \mathbf{i}^{''}) - |a_p^{'} - a_p^{''}|i_p^{'} \geq V_l.$$

As we compute only the bounds of an integer affine function, the presented method results in a greater accuracy and also in reduction of the time cost. This simple and efficient data-dependency testing method is suitable for the use in a dependence analyzer that is organized as a series of tests, progressively increasing in accuracy, as a one of the first used, simple and inexpensive tests.

Key words: data dependency, multimedia extensions, SIMD instructions, vectorizing compilers

Prejet 24. september, 2008
Odobren 8. januar, 2009

1 Uvod

Moderni splošnonamenski mikroprocesorji imajo poleg navadnega skalarnega nabora ukazov še t.i. nabor ukazov SIMD (*Single Instruction Multiple Data*) namenjen izvajanju operacij nad kratkimi vektorji oz. pakiranimi podatki. Čeprav izdelovalci mikroprocesorjev

različno poimenujejo ta nabor ukazov (npr. Intel MMX/SSE/SSE2/SSE3, Motorola Altivec, SUN VIS), gre pri vseh za tako rekoč enak nabor operacij, ki jih podpirajo ti ukazi. S temi ukazi lahko izvedemo isto operacijo nad več operandi hkrati. Operandi so praviloma 8-, 16-, 32-, ali 64-bitni in so shranjeni v registrih SIMD dolžine 128 bitov. Aritmetično-logični ukazi iz nabora ukazov SIMD ponavadi podpirajo preproste aritmetično-logične operacije (seštevanje, odštevanje, množenje, logične operacije). Npr. z enim samim ukazom SIMD za seštevanje tako lahko seštejemo dva vektorja, ki vsebujeta 16 8-bitnih komponent. Ukazi za prenos podatkov iz nabora ukazov SIMD praviloma prenašajo istoležne pomnilniške besede med pomnilnikom in registri SIMD v CPE in se od navadnih ukazov load/store razlikujejo le v dolžini prenesene besede. Nabor ukazov SIMD je predvsem namenjen hitrejšemu izvajanju nekaterih vektorskih in matričnih operacij v digitalni obdelavi signalov, ki jih opišemo s programskimi zankami (npr. množenje matrik in vektorjev, mešanje slik, DCT, FIR filtri ipd.). Pohitritev izvajanja naštetih operacij je posledica tega, da lahko z enim ukazom SIMD izvedemo isto operacijo nad več operandi ter da je treba manjkrat zajeti ukaze. Povedano drugače, programsko zanko, v kateri M-krat izvajamo isto operacijo, razvijemo N-krat ter N ukazov nadomestimo z enim samim ukazom SIMD. V praksi se izkaže, da deli programske kode, ki jih lahko zapišemo z ukazi SIMD pomenijo manjši delež v aplikacijah, zato je v praksi tipična pohitritev izvajanja programov s temi ukazi okrog 30 odstotna.

Danes je zaželjeno, da programe pišemo v visokih programskih jezikih ter z uporabo t.i. vektorizirajočih prevajalnikov dosežemo vektorizacijo zank z ukazi SIMD. To pomeni, da morajo prevajalniki analizirati programske zanke ter način dostopa do podatkov v njih. Prevajalniki morajo predvsem ugotoviti, ali sta morebiten razvoj zanke in zamenjava vrstnega reda izvajanja ukazov legalna. Odgovor na vprašanje legalnosti neke transformacije zank nam da t. i. *analiza podatkovne odvisnosti*.

Problem analize podatkovne odvisnosti v zankah, v katerih uporabljamo linearne reference na polja, se prevede na iskanje množice rešitev linearne enačbe z danimi omejitvami za posamezne spremenljivke. Ta problem je ekvivalenten problemu celoštevilskega linearnega programiranja in je torej $\mathcal{NP}$-polen problem ([10]). Danes obstaja veliko metod za iskanje celoštevilskih rešitev sistema podatkovne odvisnosti ([10]). Razvrstimo jih v dve skupini. V prvi so t. i. *približne*, v drugi pa *natančne* metode. Značilnost približnih testov je, da na dokaj preprost način poskušajo dokazati neobstoj podatkovne odvisnosti. Če jim to ne uspe, konservativno sklepajo, da odvisnost obstaja. Najpogosteje uporabljena približna testa sta test Banerjee in test GCD ([2], [11]). Odlikuje ju nizka računska zahtevnost. Gre za konservativna testa, ki tedaj, ko ne moreta dokazati neodvisnosti, prepovedujeta nadaljnjo paralelizacijo. Po drugi strani natančne metode poiščejo vse rešitve linearne enačbe in ugotovijo, kdaj in med katerimi iteracijami pride do podatkovne odvisnosti. V literaturi so predstavljeni tudi številni natančni testi. Med njimi se največ uporablja test Omega ([7]). V najslabšem primeru je časovna zahtevnost tega testa eksponentna. Test zanesljivo ugotovi obstoj podatkovne odvisnosti.

V praksi se izkaže, da so približni testi izjemno učinkoviti, zato se prevajalniki analize podatkovne odvisnosti lotevajo hierarhično: najprej s približnimi testi poskušajo pokazati, da podatkovna odvisnost v zankah ne obstaja, in nato uporabijo natančne metode za analizo zank, pri katerih približni testi niso dali pozitivnega rezultata.

V nadaljevanju bomo videli, da tedaj, ko skušamo paralelizirati programske zanke z naborom ukazov SIMD, dovolimo obstoj podatkovne odvisnosti med programskimi stavki, ki se ne bodo izvedli vzporedno, tj. kadar je razdalja med pomnilniškimi referencami dovolj velika. To je v praksi pogosto, saj so registri SIMD relativno kratki, zato obstoječi približni testi ne morejo dati zadovoljivega odgovora. V tem članku predstavimo metodo, s katero pri približnih testih dodamo določene omejitve. Tako učinkovito filtriramo in ignoriramo podatkovne odvisnosti, ki nam ne bodo preprečile paralelizacije z ukazi SIMD.

2 Osnovni pojmi

V tem razdelku bomo podali nekaj dobro znanih pojmov, ki bodo bralcu olajšali branje članaka in razumevanje predstavljene metode.

Definicija: 1 *(I_k, I^k). Naj bo $p \in \mathbb{N}$ in $\forall k \in \{1, \ldots, p\}$ naj velja $l_k, u_k \in \mathbb{Z}$ tako, da $l_k < u_k$. Množico celih števil med l_k in u_k imenujemo interval $I_k = [l_k \ldots u_k]$. Kartezični produkt prvih k intervalov je*

$$I^k = I_1 \times I_2 \times \ldots \times I_k. \tag{1}$$

V nadaljevanju bomo p, k, u_k, l_k, I_k in I^k uporabljali, kot je definirano v definiciji 1.

Definicija: 2 *(p-gnezdena for zanka). p-gnezdena for zanka je gnezdo p for zank, kot je prikazano spodaj:*

```
1 for (i1=l1; i1<=u1; i1++) {
2   for (i2=l2; i2<=u2; i2++) {
3   ...
4     for (ip=lp; ip<=up; ip++) {
5       stavki;
6     }
7   ...
8   }
9 }
```

Spremenljivka $\mathbf{i} \in I^p$ naj pomeni vrednosti vseh zančnih indeksov $(i1, i2, \ldots, ip)$ v najbolj notranji zanki. Glavni cilj metode, ki jo bomo predstavili v članku, je, da identificiramo morebitne podatkovne odvisnosti med pomnilniškimi referencami v najbolj notranji zanki p-gnezdene for zanke. Vsak dostop do elementov nekega n-dimenzionalnega polja A, ki ga v visokih programskih jezikih ponavadi označimo z $A[t_1][t_2]...[t_n]$, je v resnici dostop do neke pomnilniške lokacije. Naslov pomnilniške lokacije se izraža kot afina funkcija, ki je podana v naslednjem izreku.

Izrek: 1 *Naj bo A n-dimenzionalno polje, kjer so meje posameznih dimenzij L_j, $U_j \in \mathbb{Z}$: $L_j \leq U_j$ za vse $1 \leq j \leq n$ in naj bo A_0 naslov prvega elementa v polju. Naslov elementa $A[t_1][t_2]...[t_n]$ izrazimo z afino funkcijo*

$$f(t_1, t_2, \ldots, t_n) = A_0 + \sum_{j=1}^{n} D_j \cdot t_j, \tag{2}$$

kjer je

$$D_j = \begin{cases} 1 & ; \quad j = n \\ \prod\limits_{k=j+1}^{n} (U_k - L_k + 1) & ; \quad 1 \leq j \leq (n-1). \end{cases} \tag{3}$$

Dokaz najdemo v [11].

V nadaljevanju bomo zapis $A(f(t_1, t_2, \ldots, t_n))$ uporabljali za označevanje elementa $A[t_1][t_2]...[t_n]$. Ponekod bomo uporabili obe notaciji, odvisno od konteksta. V nadaljevanju bomo predpostavljali, da so pri referenci na polje A v najbolj notranji zanki vsi izrazi t_j afine funkcije zančnih indeksov $\mathbf{i} \in I^p$, tj. referenca na polje A bo

$$A[z_1(\mathbf{i})][z_2(\mathbf{i})] \ldots [z_n(\mathbf{i})] \tag{4}$$

kjer so $z_1, \ldots z_n : I^p \mapsto \mathbb{Z}$ afine funkcije.

Predstavljena metoda ugotavljanja podatkovne odvisnosti bo, tako kot test Banerjee, temeljila na iskanju mejnih vrednosti celoštevilske afine funkcije. Mejne vrednosti linearne celoštevilske funkcije podaja naslednji izrek ([2], [11]):

Izrek: 2 *Naj bo $p \in \mathbb{N}$ in $\forall k \in \{1, \ldots, p\}$ naj velja $l_k, u_k \in \mathbb{Z}$, tako da $l_k < u_k$. Minimum in maksimum funkcije*

$$g : I^p \mapsto \mathbb{Z}, \quad g(\mathbf{i}) = \sum_{k=1}^{p} a_k i_k$$

sta

$$\min(g(\mathbf{i})) = \sum_{k=1}^{p} a_k^+ l_k - a_k^- u_k,$$

$$\max(g(\mathbf{i})) = \sum_{k=1}^{p} a_k^+ u_k - a_k^- l_k,$$

kjer je

$$r^+ = \begin{cases} r & r > 0 \\ 0 & r \leq 0 \end{cases} \quad \textit{in} \quad r^- = \begin{cases} 0 & r \geq 0 \\ -r & r < 0 \end{cases}$$

za vsako celo število r.

3 Problem odvisnosti podatkov

Potreben pogoj za paralelizacijo zanke (oziroma za zapis več zaporednih ukazov zanke z enim samim SIMD ukazom) je neodvisnost referenc na podatke istega polja. Ugotavljanje, ali se z dvema različnima referencama na polja istega polja lahko sklicujemo na isti element, se imenuje *problem odvisnosti podatkov* in je formalno definiran z naslednjo definicijo.

Definicija: 3 *(Problem odvisnosti podatkov) Naj bosta $A(f(z'_1(\mathbf{i}), z'_2(\mathbf{i}), \ldots, z'_n(\mathbf{i})))$ in $A(f(z''_1(\mathbf{i}), z''_2(\mathbf{i}), \ldots, z''_n(\mathbf{i})))$ dve referenci na isto polje A znotra p-gnezdene for zanke. Če obstajata dva celoštevilska vektorja $\mathbf{i}', \mathbf{i}'' \in I^p$, ki zadoščata enačbi,*

$$\begin{matrix} f(z'_1(\mathbf{i}'), & z'_2(\mathbf{i}'), & \ldots, & z'_n(\mathbf{i}')) \\ f(z''_1(\mathbf{i}''), & z''_2(\mathbf{i}''), & \ldots, & z''_n(\mathbf{i}'')), \end{matrix} \quad = \tag{5}$$

potem pravimo, da sta referenci na polje A podatkovno odvisni.

V tem delu bomo predpostavili, da sta za vse $j \in \{1, \ldots n\}$ funkciji $z'_j(\mathbf{i})$ in $z''_j(\mathbf{i})$, uporabljeni v referencah opazovanega polja A, afini funkciji svojih argumentov. Upoštevajoč trditev (3), po kateri je tudi funkcija f afina funkcija svojih argumentov, lahko enačbo (5) preoblikujemo v

$$\begin{matrix} a'_0 & + & a'_1 i'_1 & + & a'_2 i'_2 & + & \ldots & + & a'_p i'_p = \\ a''_0 & + & a''_1 i''_1 & + & a''_2 i''_2 & + & \ldots & + & a''_p i''_p, \end{matrix} \tag{6}$$

kjer so

$$a'_0, a''_0, a'_1, a''_1, \ldots, a'_p, a''_p \in \mathbb{Z}$$

izraženi iz celoštevilskih argumentov funkcij f in $z'_1, z''_1, z'_2, z''_2, \ldots, z'_n, z''_n$.

4 Ugotavljanje podatkovne odvisnosti

V nadaljevanju se bomo osredotočili na p-gnezdeno zanko in iskali podatkovno odvisnost referenc na dano polje A. Predpostavili bomo, da so v polju A elementi istega tipa in da se referenci na elemente tega polja v dveh zaporednih iteracijah najbolj notranje zanke razlikujeta za 1 ali -1 (enotski korak). Te predpostavke izvirajo iz dejstva, da lahko tedaj, ko razpolagamo z registrom SIMD velikosti

$V_l b$, z enim ukazom SIMD izvedemo operacijo nad V_l podatki dolžine b, le, če so ti podatki zaporedno shranjeni v pomnilniku. Vprašanje, s katerim se bomo ukvarjali v nadaljevanju članka, je, ali (oziroma, kdaj) lahko razvijemo V_l korakov najbolj notranje zanke in jih zamenjamo z enim samim ukazom SIMD. Predstavili bomo metodo, ki analiziran podatkovno odvisnost referenc na dano polje in odgovori na zgornje vprašanje – pritrdilno, kadar je tako razvitje mogoče, in nikalno, kadar ni oziroma kadar zanesljivega odgovora ne poznamo. Naša metoda, ki upošteva dejstvo, da je za paralelizacijo z ukazi SIMD odločilnega pomena, da je razdalja med posameznimi referencami na polje v najbolj notranji zanki večja ali kvečjemu enaka velikosti registra SIMD V_l dopolnjuje znani test Banerjee, ki o neodvisnosti podatkov presoja zgolj na podlagi ocene intervala, znotraj katerega se lahko giblje preiskovana referenca. Test z omenjeno dopolnitvijo razširja uporabnost in učinkovitost Banerjeevega testa za SIMD prevajalnike. Pred predstavitvijo metode zapišimo še nekaj oznak in definicij.

Naj bosta $\mathbf{i}' = (i_1', i_2', \ldots i_p') \in I^p$ in $\mathbf{i}'' = (i_1'', i_2'', \ldots i_p'') \in I^p$. Če definiramo

$$\begin{aligned}\zeta(\mathbf{i}', \mathbf{i}'') = \; & a_0'' + a_1'' i_1'' + \ldots + a_{p-1}'' i_{p-1}'' \\ & - a_0' - a_1' i_1' - \ldots - a_{p-1}' i_{p-1}',\end{aligned}$$

potem splošna enačba odvisnosti (6) postane

$$a_p' i_p' - a_p'' i_p'' = \zeta(\mathbf{i}', \mathbf{i}''). \tag{7}$$

Razdaljo $d(\mathbf{i}', \mathbf{i}'')$ med referencami na elemente polja v najbolj notranji zanki, ki je v [11] definirana kot

$$d(\mathbf{i}', \mathbf{i}'') = i_p'' - i_p' \tag{8}$$

lahko izrazimo iz (7) kot

$$\begin{aligned} d(\mathbf{i}', \mathbf{i}'') \; & = i_p'' - i_p' = \\ & = (a_p' - 1) i_p' - (a_p'' - 1) i_p'' - \zeta(\mathbf{i}', \mathbf{i}''). \end{aligned} \tag{9}$$

Paralelizacijo Banerjee lahko izvedemo, če je razdalja med posameznimi referencami večja ali kvečjemu enaka velikosti registra SIMD V_l (t.j. številu podatkov, nad katerimi bomo izvajali operacijo v enem ukazu SIMD) oziroma kadar velja:

$$d(\mathbf{i}', \mathbf{i}'') \leq 0 \quad \text{ali} \quad d(\mathbf{i}', \mathbf{i}'') \geq V_l \tag{10}$$

za vse $(\mathbf{i}', \mathbf{i}'') \in I^p$. Upoštevajoč dejstva, da se lahko indeks najbolj notranje for zanke spreminja le za ± 1 in da smo se omejili na enotski korak ($|a_p'| = |a_p''| = 1$), lahko predstavimo izrek, na katerem temelji naša metoda za ugotavljanje podatkovne odvisnosti.

Izrek: 3 *Naj bo $\zeta : I^p \times I^p \mapsto \mathbb{Z}$ afina funkcija s celoštevilskimi koeficienti in naj bo problem odvisnosti podatkov v najbolj notranji zanki p-gnezdene for zanke z mejami $l_p \leq i_p', i_p'' \leq u_p$ predstavljen z enačbo $a_p' i_p' - a_p'' i_p'' = \zeta(\mathbf{i}', \mathbf{i}'')$. Če je $|a_p'| = |a_p''| = 1$, potem je razdalja med referencami $d(\mathbf{i}', \mathbf{i}'')$ enaka*

$$d(\mathbf{i}', \mathbf{i}'') = -a_p'' \zeta(\mathbf{i}', \mathbf{i}'') - |a_p' - a_p''| i_p'. \tag{11}$$

Dokaz: 1 *Pokazali bomo, da je vrednost razdalje $d(\mathbf{i}', \mathbf{i}'')$, izračunane po definiciji (9)*

$$d(\mathbf{i}', \mathbf{i}'') = (a_p' - 1) i_p' - (a_p'' - 1) i_p'' - \zeta(\mathbf{i}', \mathbf{i}'').$$

in po formuli (11)

$$d(\mathbf{i}', \mathbf{i}'') = -a_p'' \zeta(\mathbf{i}', \mathbf{i}'') - |a_p' - a_p''| i_p',$$

ki jo predlaga izrek, enaka v vseh (štirih) primerih:

1. $a_p' = a_p'' = 1$
   - *po (9):* $d(\mathbf{i}', \mathbf{i}'') = (1-1)i'p - (1-1)i_p'' - \zeta(\mathbf{i}', \mathbf{i}'') = -\zeta(\mathbf{i}', \mathbf{i}'')$
   - *po (11):* $d(\mathbf{i}', \mathbf{i}'') = -1\zeta(\mathbf{i}', \mathbf{i}'') - |1-1| i_p' = -\zeta(\mathbf{i}', \mathbf{i}'')$.
2. $a_p' = a_p'' = -1$
   - *po (9):* $d(\mathbf{i}', \mathbf{i}'') = -2i_p' + 2i_p'' - \zeta(\mathbf{i}', \mathbf{i}'') = 2(i_p'' - i_p') - \zeta(\mathbf{i}', \mathbf{i}'') = 2d(\mathbf{i}', \mathbf{i}'') - \zeta(\mathbf{i}', \mathbf{i}'') \Rightarrow d = \zeta(\mathbf{i}', \mathbf{i}'')$
   - *po (11):* $d(\mathbf{i}', \mathbf{i}'') = -(-1)\zeta(\mathbf{i}', \mathbf{i}'') - |-1-(-1)| i_p' = \zeta(\mathbf{i}', \mathbf{i}'')$.
3. $a_p' = 1, a_p'' = -1$
   - *po (9):* $d(\mathbf{i}', \mathbf{i}'') = (1-1)i_p' - (-1-1)i_p'' - \zeta(\mathbf{i}', \mathbf{i}'') = 2i_p'' - \zeta(\mathbf{i}', \mathbf{i}'')$. *Po definiciji velja* $\zeta(\mathbf{i}', \mathbf{i}'') = a_p' i_p' - a_p'' i_p'' = i_p' + i_{p''}''$, *zato* $i_p'' = \zeta(\mathbf{i}', \mathbf{i}'') - i_p'$ *in* $d(\mathbf{i}', \mathbf{i}'') = 2i_p'' - \zeta(\mathbf{i}', \mathbf{i}'') = 2\zeta(\mathbf{i}', \mathbf{i}'') - 2i_p' - \zeta(\mathbf{i}', \mathbf{i}'') = \zeta(\mathbf{i}', \mathbf{i}'') - 2i_p'$.
   - *po (11):* $d(\mathbf{i}', \mathbf{i}'') = -(-1)\zeta(\mathbf{i}', \mathbf{i}'') - |1-(-1)| i_p' = \zeta(\mathbf{i}', \mathbf{i}'') - 2i_p'$.
4. $a_p' = -1, a_p'' = 1$
   - *po (9):* $d(\mathbf{i}', \mathbf{i}'') = (-1-1)i_p' - (1-1)i_p'' - \zeta(\mathbf{i}', \mathbf{i}'') = -2i_p' - \zeta(\mathbf{i}', \mathbf{i}'')$
   - *po (11):* $d(\mathbf{i}', \mathbf{i}'') = -1\zeta(\mathbf{i}', \mathbf{i}'') - |-1-1| i_p' = -2i_p' - \zeta(\mathbf{i}', \mathbf{i}'')$.

□

Izrek 3 predstavlja preprosto in poceni metodo za ugotavljanje podatkovne odvisnosti za procesorje z ukazi SIMD: najbolj notranjo zanko p-gnezdene for zanke lahko razvijemo v zaporedje ukazov SIMD, če velja ena od spodnjih neenakosti:

$$\max_{i',i''\in I^p} -a''_p\zeta(\mathbf{i}^{'},\mathbf{i}^{''}) - |a'_p - a''_p|i'_p \leq 0 \qquad (12)$$

ali

$$\min_{i',i''\in I^p} -a''_p\zeta(\mathbf{i}^{'},\mathbf{i}^{''}) - |a'_p - a''_p|i'_p \geq V_l. \qquad (13)$$

Izrek 3 in neenačbi (12) ter (13) uporabimo v naslednjem algoritmu:

Algoritem: 1 *Algoritem vrne TRUE, kadar so reference na elemente polja neodvisne, in FALSE, kadar podatkovna odvisnost morda obstaja.*

1. *s pomočjo izreka 1 zapiši afini funkciji potencialno problematičnih (odvisnih) referenc na elemente polja znotraj najbolj notranje zanke,*
2. *s pomočjo izreka 3 izrazi funkcijo* $d(\mathbf{i}',\mathbf{i}'')$,
3. *s pomočjo izreka 2 poišči*
   $m = \min_{i',i''\in I^p} d(\mathbf{i}^{'},\mathbf{i}^{''})$
   in
   $M = \max_{i',i''\in I^p} d(\mathbf{i}^{'},\mathbf{i}^{''})$
4. **če** *((*$M \leq 0$*)* **ali** *(*$m \geq V_l$*))*
   vrni *TRUE;*
   sicer
   vrni *FALSE.*

5 Tehnične podrobnosti algoritma

Naj bo A n-dimenzionalno polje z mejami za posamezno dimenzijo L_j, $U_j \in \mathbb{Z} : L_j \leq U_j$ za vse $1 \leq j \leq n$ in naj A_0 pomeni naslov prvega elementa tega polja. Kot neposredna posledica izreka 1 lahko koeficiente D_j izračunamo z

$$\begin{array}{l} D_n\text{=1;} \\ \texttt{for}\ (i = n-1; i >= 1; i--) \\ \quad D_i = D_{i+1} * (U_{i+1} - L_{i+1} + 1); \end{array} \qquad (14)$$

Naj bosta $A(f(z'_1(\mathbf{i}), z'_2(\mathbf{i}), \ldots, z'_n(\mathbf{i})))$ in $A(f(z''_1(\mathbf{i}), z''_2(\mathbf{i}), \ldots, z''_n(\mathbf{i})))$ dve potencialno problematični (odvisni) referenci na elementa polja A znotraj p-gnezdene for zanke in naj bo

$$z'_j(\mathbf{i}') = z'_{j0} + \sum_{r=1}^{p} z'_{jr} i'_r$$

in

$$z''_j(\mathbf{i}'') = z''_{j0} + \sum_{r=1}^{p} z''_{jr} i''_r,$$

za vse $j = 1, 2, \ldots n$. Po izreku 1 je

$$f(z_1(\mathbf{i}'), \ldots, z_n(\mathbf{i}')) = A_0 + \sum_{j=1}^{n} D_j z'_j(\mathbf{i}'),$$

in zato

$$\begin{aligned} f(z_1(\mathbf{i}'), \quad \ldots \quad , z_n(\mathbf{i}')) = \\ = \quad A_0 + \sum_{j=1}^{n} D_j (z_{j0} + \sum_{r=1}^{p} z'_{jr} i'_r) = \\ = \quad A_0 + \sum_{j=1}^{n} D_j z'_{j0} + \sum_{r=1}^{p} (\sum_{j=1}^{n} D_j z'_{jr}) i'_r. \end{aligned}$$

$$\begin{aligned} a'_0 &= A_0 + \sum_{j=1}^{n} D_j z'_{j0} \\ a'_r &= \sum_{j=1}^{n} D_j z'_{jr} \text{ za vse } r = 1, \ldots p \end{aligned} \qquad (15)$$

Podobno velja za a''_0 in a''_r.
Afino funkcijo $d(\mathbf{i}^{'},\mathbf{i}^{''})$ zapišimo kot

$$d(\mathbf{i}^{'},\mathbf{i}^{''}) = d'_0 + \sum_{r=1}^{p} d'_r i'_r + d''_0 + \sum_{r=1}^{p} d''_r i''_r.$$

Po izreku (3) je $d(\mathbf{i}^{'},\mathbf{i}^{''}) = -a''_p\zeta(\mathbf{i}^{'},\mathbf{i}^{''}) - |a'_p - a''_p|i'_p$, zato so skoraj vsi koeficienti $d(\mathbf{i}^{'},\mathbf{i}^{''})$ produkt ustreznega koeficienta a' (oziroma a'') v $\zeta(\mathbf{i}^{'},\mathbf{i}^{''})$ in a''_p (oziroma $-a''_p$). Izjema sta le koeficient d'_p, ki je enak $-|a'_p - a''_p|$, in d''_p, ki je enak 0.

$$\begin{aligned} d'_r &= +a''_p a'_r \quad \text{for all } r = 0, \ldots p-1 \\ d'_p &= -|a'_p - a''_p| \\ d''_p &= 0 \\ d''_r &= -a''_p a''_r \quad \text{for all } r = 0, \ldots p-1 \end{aligned} \qquad (16)$$

Na koncu uporabimo še izrek (2) in izračunamo najmanjšo (m) in največjo (M) vrednost funkcije $d(\mathbf{i}^{'},\mathbf{i}^{''})$:

$$\begin{aligned} m &= d'_0 + d''_0 + \sum_{k=1}^{p} e_k^+ l_k - e_k^- u_k \\ M &= d'_0 + d''_0 + \sum_{k=1}^{p} e_k^+ u_k - e_k^- l_k, \end{aligned} \qquad (17)$$

pri čemer je e_k okrajšava za $d'_k + d''_k$.

6 Rezultati in sklep

V tem delu smo predstavili preprosto, a učinkovito metodo za ugotavljanje podatkovne neodvisnosti referenc na elemente polja znotraj vgnezdene for zanke. Metoda se lahko uporablja kot pripomoček za paralelizacijo kode na procesorjih SIMD.

Znane metode za ugotavljanje neodvisnosti temeljijo na iskanju celoštevilskih rešitev enačbe, ki opisuje problem odvisnosti – če tako rešitev najdejo, metode predpostavijo podatkovno odvisnost (čeprav ta morda ne obstaja). Tak konservativen pristop je še posebno neučinkovit pri uporabi na procesorjih SIMD, saj so registri SIMD relativno kratki in omogočajo paralelizacijo marsikatere zanke, katere vektorizacija na tradicionalnih vektorskih procesorjih ni mogoča.

Metodo smo testirali na knjižnici LAPACK (BLAS) in na veliki množici sintetičnih testov. Skupno smo analizirali nekaj več kot 3×10^8 zank z dostopom MIV (Multiple Index Variable) [1] do dvo-dimenzionalnih polj. Testi so pokazali, da predstavljena metoda odkrije med 1-2% več podatkovno neodvisnih dostopov kot test Banerjee (višji procent izboljšanja smo dosegli pri večjih poljih). Rezultati so primerljivi tudi z rezultati iz [8], kjer so ugotovili, da test Omega odkrije le 5% več podatkovno neodvisnih dostopov kot test Banerjee, vendar na račun bistevno povečane računske zahtevnosti. Kljub tem, da je časovna zahtevnost predlagane metode linearna, tj. enaka kot pri Banerjeejevem testu, dosežemo nezanemarljivo izboljšanje pri filtriranju podatkovnih odvisnosti, ki ne prepovedujejo vektorizacije SIMD.

Namesto iskanja celoštevilskih rešitev enačbe podatkovne odvisnosti naša metoda temelji na določanju najmanjše in največje razdalje med potencialno problematičnima referencama znotraj iteracijskega prostora. Tak pristop omogoča večjo natančnost, hkrati pa je metoda časovno nezahtevna, saj izračuna le meje celoštevilske afine funkcije. Metoda je primerna za uporabo v analizatorju odvisnosti, ki je organiziran kot zaporedje testov z naraščajočo zanesljivostjo, kot eden prvih, preprostih in poceni testov.

7 Literatura

[1] Allen R., Kennedy K. *Optimizing Compilers for Modern Architectures: A Dependence-based Approach*. Morgan Kaufmann Publishers Inc., 2001.

[2] Banerjee U. *Dependence Analysis: A Book Series on Loop Transformations for Restructuring Compilers*. Kluwer Academic Publishers, Dordrecht, 1997.

[3] Bik A.J.C., Girkar M., Grey P.M., Tian X.M. Automatic Intra-Register Vectorization for the Intel (R) Architecture. *International Journal of Parallel Programming*. Vol 30., No. 2, pp. 65-98. 2002.

[4] Huang T.C., Yang C.M. Data Dependence Analysis for Array References. *The Journal of Systems and Software*. No. 52, pp. 55-65, 2000.

[5] Krall A., Lelait S. Compilation Techniques for Multimedia Processors. *International Journal of Parallel Programming*. Vol. 28, No. 4, pp. 347-361, 2000.

[6] Muchnick S. Advanced Compiler Design & Implementation . Morgan Kauffman Publishers, 1997.

[7] Pough W. A Practical Algorithm for Exact Array Dependence Analysis. *Communications of the ACM*. Vol. 35, No. 8, pp. 102-114, 1992.

[8] Psarris K., Klappholz D., Kong X. An Experimental Evaluation of Data Dependence Analysis Techniques *IEEE Transactions on Parallel and Distributed Systems*. Vol. 15, No. 3, pp. 196-213, March 2004.

[9] Sreraman N., Govindarajan R. A Vectorizing Compiler for Multimedia Extensions. *International Journal of Parallel Programming*. Vol. 28, No. 4, pp. 363-400, 2000.

[10] Wolfe M.J., Tseng C.W. The Power Test for Data Dependence. *IEEE Transactions on Parallel and Distributed Systems*. Vol. 3, No. 5, pp. 591-601, September 1992.

[11] Zima H.P., Chapman B.M. *Supercompilers for Parallel and Vector Computers*. Addison-Wesley Publishing Company, 1990.

Patricio Bulić, diplomirani inženir elektrotehnike, je doktoriral leta 2004 na Fakulteti za računalništvo in informatiko v Ljubljani, kjer je trenutno zaposlen. Je član Laboratorija za računalniško arhitekturo. Njegovo raziskovalno delo obsega računalniške arhitekture, paralelno procesiranje ter načrtovanje digitalnih in vgrajenih sistemov.

Tomaž Dobravec, diplomiran matematik, je doktoriral leta 2004 na Fakulteti za računalništvo in informatiko v Ljubljani, kjer je trenutno tudi zaposlen. Raziskovalno se v okviru Laboratorija za algoritme in podatkovne strukture ukvarja z razvojem in analizo algoritmov, največ pozornosti pa namenja omrežjem s simetrično topologijo in zaščiti podatkov.