PARALELNÍ DIFERENCIÁLNÍ EVOLUCE PRO SYSTÉMY S ARCHITEKTUROU SIMD

Jiří Bůžek[1]
[1]*katedra informatiky a počítačů, Přírodovědecká fakulta, Ostravská univerzita v Ostravě, jiri.buzek@osu.com*

Abstrakt

Cílem toho článku je představení návrhu paralelní implementace algoritmu diferenciální evoluce pro výpočetní systémy s architekturou SIMD. Implementace algoritmu byla otestována v prostředí simulujícím chování systémů SIMD. Výstupy navržené implementace byly porovnány s existující sekvenční variantou algoritmu.

Klíčová slova: *Diferenciální evoluce; paralelismus.*

Úvod

Mějme účelovou funkci $f: D \to \mathbb{R}$; $-\infty < f(x) < \infty, \forall x \in D, D \subseteq \mathbb{R}^d$, kde D je prohledávaný prostor a $d \in \mathbb{N}$ je dimenze úlohy. Úkolem globální optimalizace je najít takový bod, který je globálním minimem nebo maximem účelové funkce. Pro řešení úlohy nalezení globálního minima máme najít všechny body $x^* \in D$, pro které platí, že $f(x^*) \leq f(x)$, pro $\forall x, x \in D$. Přitom předpokládejme, že D je souvislá ohraničená oblast $D = \prod_{i=1}^{d} < a_i, b_i >$, $a_i < b_i, i = 1,2, \dots, d$, kde a_i je horní a b_i je horní hranice.

Diferenciální evoluce, dále jen DE, byla vyvinuta v roce 1995 Stornem a Pricem [3]. Projevila se jako robustní a spolehlivá a stala se velmi používaným evolučním algoritmem pro řešení problému spojité globální optimalizace. DE pracuje s populací Np d-rozměrných vektorů. Zkusmé vektory, kandidáti na řešení, jsou vytvářeny procesy mutace a křížení. Zkusmý vektor je považován za úspěšný v případě, že jeho hodnota vyhodnocené účelové funkce je menší nebo rovna hodnotě účelové funkce současného jedince. Takový úspěšný vektor nahrazuje současného jedince v další populaci. DE postupně generuje nové populace, dokud nejsou splněny ukončovací podmínky.

Single Instruction / Multiple Data, zkráceně SIMD, technologie je široce používaná v oblastech 3D grafiky, zpracovávání obrazu, zvuku atd. Použitím SIMD výpočetních metod jsme schopni souběžně zpracovat množinu dat jednou instrukcí. Například v konvenčních systémech MIMD, SISD musí být součet dvou vektorů realizován postupně složku po složce, kdežto SIMD jsou schopny tuto operaci provést naráz jednou instrukcí se stejným výsledkem.

Cílem tohoto článku je představení návrhu implementace paralelní DE pro výpočetní systémy s architekturou SIMD.

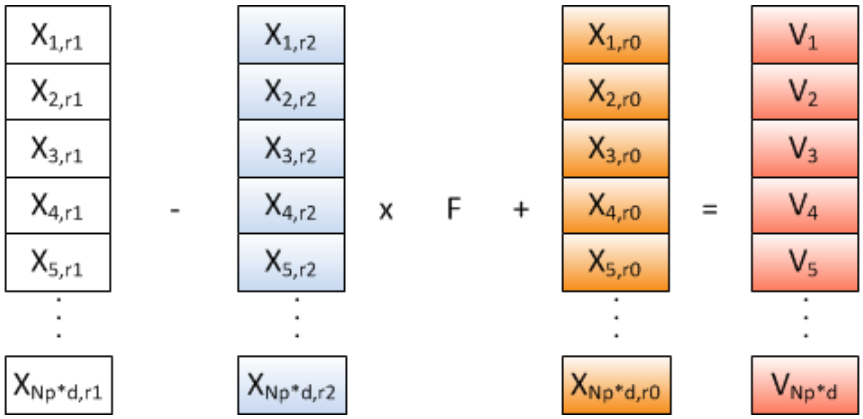
Návrh implementace

Algoritmus DE musí být dekomponován do co nejdelších bloků posloupností instrukcí, abychom se vyhnuli režiím spojených s předáváním řízení kontroly algoritmu mezi výpočetními jednotkami. Populaci v paměti reprezentuje jednorozměrné pole reálných čísel. Nad jednotlivými prvky jsou prováděny navržené posloupnosti instrukcí.

Následující rovnice znázorňuje vytvoření vektoru diferenciální mutaci:

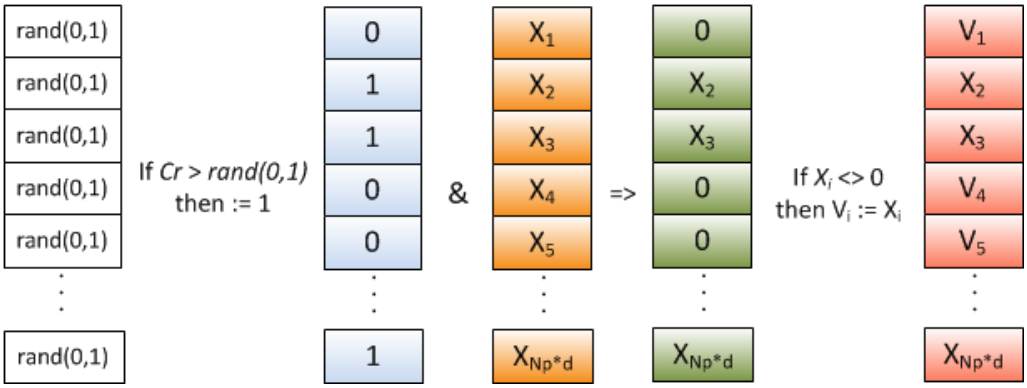
$$v_i = x_{r0} + F\ (x_{r1} - x_{r2}).$$

Index *i* je index současného vektoru, jehož hodnoty mají být nahrazeny. $F \in (0,1)$ je parametr udávající míru, s jakou se populace vyvíjí. Indexy vektorů *r0, r1 a r2* staré populace jsou náhodně vybrány za podmínky $r0 \neq r1 \neq r2 \neq i$. Obrázek 1 ilustruje posloupnost SIMD operací, které vytvoří přechodnou populaci vektorů mutací. Před provedením této posloupnosti jsou paralelně vygenerovány indexy *r0, r1 a r2* pro každého jedince. Generování náhodných čísel se věnuje následující kapitola.



Obrázek 1. posloupnost SIMD operací pro diferenciální mutaci.

Nový vektor u_i (zkusmý vektor) vznikne křížením vektoru x_i a v_i. V této práci je použito binomické křížení. Binomické křížení je určeno následujícím pravidly: $u_i = v_{i,j}$ když $rand(0,1) \leq Cr$ nebo $j = j_{rand}$, $u_i = x_{i,j}$ když $rand(0,1) > Cr$, kde j_{rand} je náhodně vybraný index složky z $0,1,\ldots,d-1$, $rand(0,1)$ je náhodné reálné číslo z intervalu [0,1), *Cr* je vstupní parametr regulující míru křížení a index *j* je konkrétní složka vektoru. V implementaci pro SIMD se nejdříve vygeneruje sekvence náhodných reálných čísel o délce $Np * d$. Nad všemi náhodnými prvky se paralelně provede operace porovnání, podle podmínky $rand(0,1) > Cr$. Při splnění této podmínky se v dočasném poli stejné délky nastaví příznaky na hodnotu true. Pro podmínku $j = j_{rand}$ se operace provedou podobně. Podle příznaků nastavených na hodnotu true se paralelně provede operace přiřazení prvků ze staré populace do přechodné. Tímto jsou vygenerovány všechny zkusmé vektory. Obrázek 2 ilustruje křížení provedené posloupností SIMD operací pro podmínku $rand(0,1) > Cr$.



Obrázek 1. posloupnost SIMD operací pro křížení.

Po vyhodnocení účelové funkce je provedena selekce. Pokud hodnota účelové funkce vektoru u_i, je menší nebo rovna hodnotě účelové funkce vektoru x_i, pak vektor u_i nahradí vektor x_i v další generaci. Selekci lze provést souběžně jako porovnání funkčních hodnot staré populace a přechodné populace. Dále lze také paralelně provést hromadné nahrazení prvků staré populace prvky přechodné.

Generování pseudonáhodných čísel

Aby byla paralelní implementace efektivní, je potřeba také paralelně generovat sekvence náhodných čísel, které jsou potřebné v procesech mutace a křížení. Jako generátor náhodných čísel je použit lineární kongruentní generátor [1], dále jen LCG. LCG je definován následujícím vztahem:

$$x_{i+1} = (ax_i + c)\ mod\ m,$$

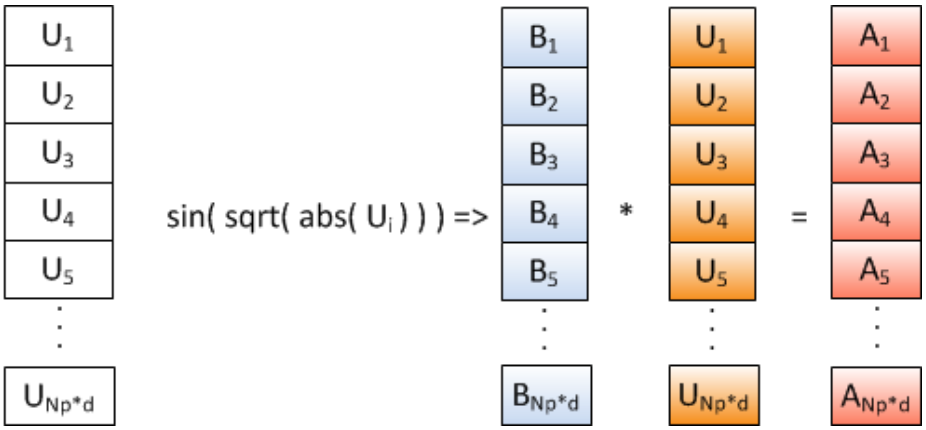
kde *a, c, m* jsou konstanty zvolené podle [2] a x_i je současný stav. Následující *k*-tý stav generátoru se vypočítá podle vztahu:

$$x_{i+k} = (a^k x_i + (a^k - 1) * c\ /\ b)\ mod\ m, k \geq 0, i \geq 0, b = a - 1$$

Pro získání sekvence *m* náhodných čísel jsou při inicializaci algoritmu vypočtena všechna $a^k, k = 0,1, \ldots, m-1$. Podle předchozího vztahu můžeme paralelně vygenerovat náhodné čísla dosazením předem vypočtených mocnin konstanty *a*.

Účelové funkce

Výpočet hodnoty účelové funkce musí také přizpůsoben architektuře SIMD, pokud chceme dosáhnout co největšího výkonu. Pro otestování paralelní implementace byl vybrán optimalizační problém, funkce Schwefel[4] daná vztahem $f(x) = 418.9829d - \sum_{i=1}^{d} x_i \sin(\sqrt{|x_i|})$. Pro SIMD je nejdříve proveden výpočet $x_i \sin\left(\sqrt{|x_i|}\right)$ nad všemi prvky pole. Posloupnost SIMD operací pro tento výpočet ilustruje obrázek 3.



Obrázek 1. SIMD operace pro $x_i \sin\left(\sqrt{|x_i|}\right)$.

Součet přes všechny *i* lze optimalizovat rozdělením dat do segmentů, paralelně vypočíst mezisoučty a výsledky těchto mezisoučtů následně sekvenčně sečíst.

Experimentální testování

Paralelní algoritmus byl implementován v prostředí .NET v programovacím jazyce C#. Chování architektury SIMD bylo simulováno postupným vykonáváním navržených posloupností operací nad jednotlivými prvky polí cyklem for. Výstupy paralelní implementace byly porovnány s výstupy existující sekvenční varianty DE. U obou variant byl použit stejný generátor náhodných čísel LCG inicializovaný na stejnou hodnotu jak pro sekvenční, tak pro paralelní variantu. To znamená, že oba generátory generovaly totožné sekvence náhodných čísel. Celkem bylo provedeno třicet běhů pro funkci Schwefel, pro různé inicializační hodnoty generátorů. Výstupy sekvenční a paralelní implementace se vzájemně nelišily.

Závěr

V této práci byl navržen paralelní algoritmus diferenciální evoluce pro systémy s architekturou SIMD. Navržený algoritmus byl implementován v prostředí .NET v programovacím jazyce C#. Paralelní implementace byla otestována v prostředí simulujícím zpracování operací SIMD. Výstupy paralelní implementace byly porovnány s existující sekvenční variantou. Výsledky se vzájemně nelišily. V budoucnu bude navržený algoritmus přizpůsoben hardwarovým požadavkům a bude implementován na architektuře CUDA nebo na instrukční sadě SSE procesorů Intel.

Poděkování

Rád bych touto formou poděkoval panu doc. Ing. Josefu Tvrdíkovi, CSc. za jeho cenné rady, nápady, připomínky a odborné vedení. Tento článek vznikl v rámci řešení projektu Studentské grantové soutěže (SGS15/PřF/2015), který je podporován Ministerstvem školství, mládeže a tělovýchovy.

Literatura

[1.] Knuth, D.E. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, Second edition*. Addision-Wesley, Massachusetts, second edition, 1981.

[2.] L'ecuyer, P. *Good parameters and implementations for combined multiple recursive random number generators*. Operations Research. 1999, 47, 1, s. 159-164.

[3.] Storn, R. Price K. *Differential evolution-a simple and effcient heuristic for global optimization over continuos spaces*. Journal of global optimization, 11(4):341-359, 1997.

[4.] Tvrdík, J. *Stochastické algoritmy pro globální optimalizaci*. Ostravská univerzita v Ostravě, 2010.

Abstract

This article introduces a parallel implementation of Differential evolution for systems with SIMD architecture. Proposed implementation was tested on special environment which is simulating SIMD behavior. Immediate outputs of proposed implementation were compared with outputs of existing sequential algorithm.