E.H. HOVHANNISYAN

EVALUATING THE TEST ASSIGNMENT QUALITY

At testing, there is a possibility of a wrong estimation of the trainee's knowledge in case of the random correct answer input. Different types of test assignments have been evaluated from the standpoint of the magnitude of such probability. A model for estimating the probability of guessing the correct answers of the test assignments at adaptive testing is proposed.

Keywords: classification of different types of test assignments, probability estimation.

ՀՏԴ 004.4:519.688

Ն.Հ. ԱԲՐՈՅԱՆ, Ռ.Գ. ՀԱԿՈԲՅԱՆ

ՏԵՍԱԿԱՎՈՐՄԱՆ ԱԼԳՈՐԻԹՄՆԵՐԻ ՁՈՒԳԱՀԵՌԱՑՄԱՆ ՄԵԹՈԴՆԵՐԻ ՀԵՏԱԶՈՏՈՒՄԸ ԵՎ ՄՇԱԿՈՒՄԸ

Հետազոտվել, մշակվել և փորձարկվել են տեսակավորման տարբեր ալգորիթմների զուգահեռացման մի քանի եղանակներ: Ցույց է տրվում, թե ինչպես կարելի է զուգահեռացման միջոցով բարելավել ալգորիթմի (ինչպիսին են, օրինակ, պղպջակային, միաձուլման, արագ տեսակավորման ալգորիթմները) արագագործությունը և համեմատել այն իր հաջորդական տարբերակի հետ:

Առանցքային բառեր. տեսակավորման ալգորիթմներ, պղպջակային տեսակավորում, միաձուլման տեսակավորում, արագ տեսակավորում, հոսքեր, արագացում:

Ներածություն: 1986-2002թթ. միկրոպրոցեսորների արագագործությունն աճել է տարեկան միջինում 50%-ով: Իսկ 2002թ. սկսած՝ մեկ միջուկով պրոցեսորների արագագործությունն աճել է տարեկան միջինում 20%-ով: Իհարկե, այս տարբերությունն ահռելի է: Մեկ միջուկանի պրոցեսորների արագագործության փաստացի աճի կանգնելու պատճառով էլ 2005թ. սկսած միկրոպրոցեսորների հիմնական արտադրողները որոշեցին առանձին պրոցեսորի արագագործությունը բարձրացնելու փոխարեն տեղադրել մեկից ավել պրոցեսորներ: Հենց դա էլ, փաստացիորեն, եղավ ժամանակակից բազմամիջուկ պրոցեսորների սկիզբը [1]: Այդ փոփոխությունը բավականին կարևոր փոփոխություն էր ծրագրային ապահովման ոլորտում: Ուղղակիորեն միջուկների քանակն ավելացնելով՝ ծրագրերի արագագործությունը չի կարող բարելավել: Շատ ծրագրեր հաղորդակցված չեն մի քանի պրոցեսորային միջուկների գոյությանը, ուստի նրանց արագագործությունը մեկ կամ բազմամիջուկային պրոցեսորների դեպքում գործնականում կլինի գրեթե նույնը [1]: Նման դեպքում ծրագրի արագագործությունը բարձրացնելու համար անհրաժեշտ է զուգահեռացնել նրա աշխատանքը, ավելի կոնկրետ՝ զուգահեռացնել այն ալգորիթմները, որոնք օգտա-

գործվում են տվյալ ծրագրում: Ծրագրային ապահովման մեջ մշտապես լայն տարածում են ունեցել և ունեն տեսակավորման ալգորիթմները: Համակարգչային գիտության շատ գիտնականներ տեսակավորման ալգորիթմները համարում են հիմնարար խնդիրներ ալգորիթմների ոլորտում: Ներկայումս հայտնի են տեսակավորման բազմաթիվ ալգորիթմներ, որոնք միմյանցից տարբերվում են ֆունկցիոնալությամբ, արագագործությամբ, ռեսուրսների օգտագործմամբ և այլն: Գործնական տարբեր խնդիրներում առավել արդյունավետ կարող են լինել տարբեր ալգորիթմներ՝ կախված համակարգչի հիշողության հիերարխիայից, ծրագրային միջավայրից և այլն [2]: Այս աշխատանքում որպես օրինակ դիտարկվում են պղպջակային, միաձուլման, արագ տեսակավորման ալգորիթմները: Մեր նպատակն է հետազոտել, փորձարկել և առաջարկել տեսակավորման ալգորիթմների զուգահեռացման մեթոդներ, որոնց արագագործությունը կլինի ավելի բարձր՝ համեմատած իրենց նախնական տարբերակներին: Այսպիսով, զուգահեռացնելով տեսակավորման ալգորիթմները, փաստացիորեն կբարձրանա շատ ծրագրերի արդյունավետությունը:

Ծրագրի աշխատանքի արագագործության աճը գնահատելու համար նախ՝ անհրաժեշտ է սահմանել որոշակի չափանիշ, որը կոչվում է արագացում: Վերջինս ցույց է տալիս, թե զուգահեռացված ծրագիրը քանի անգամ է ավելի արագ կատարվում իր հաջորդական տարբերակից, եթե երկուսն էլ լուծում են միևնույն խնդիրը [1]: Եթե T_s-ը հաջորդական ծրագրի կատարման ժամանակն է, իսկ T_p–ն՝ զուգահեռացված ծրագրի կատարման ժամանակը, ապա արագացման բանաձևը կլինի՝

$$S = \frac{T_S}{T_P}; \qquad (1)$$

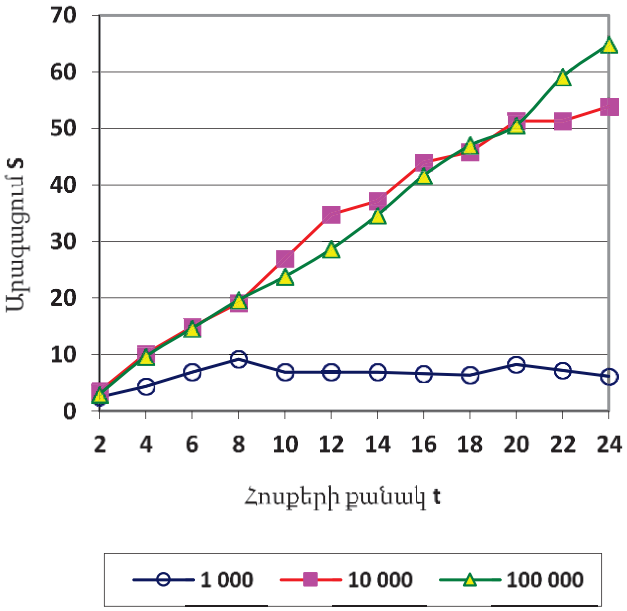
Եթե ծրագիրն աշխատում է p միջուկների օգտագործմամբ, ապա լավագույն դեպքում ՝

$$T_S = \mathrm{T_P} \cdot p:$$

Սա կոչվում է գծային արագացում, բայց համաձայն Ամդալի օրենքի՝ նույնիսկ կատարյալ զուգահեռ ծրագրում շատ դժվար է ստանալ նման արագացում, քանի որ յուրաքանչյուր ծրագրում գոյություն ունի որոշակի α կոտորակային մաս, որը չի կարող զուգահեռացվել: Այս աշխատանքում ծրագրի զուգահեռացման արագացումը կհաշվվի ըստ (1) բանաձևի:

Զուգահեռացում հոսքերի միջոցով: Հոսքերը հնարավորություն են տալիս ծրագրերը բաժանել առավելապես իրարից անկախ առաջադրանքների այնպես, որ եթե հոսքերից մեկը ինչ-որ պատճառով կանգ է առել, ապա մյուսները կշարունակեն իրենց աշխատանքը [1]: Կարող են լինել հոսքերի միջոցով զուգահեռացման

տարբեր եղանակներ: Վերջիններս կարող են լինել թե՛ ընդհանուր մոտեցումով, թե՛ հատուկ առանձին ալգորիթմին: Որպես ընդհանուր եղանակ կարող է լինել հետևյալը. մուտքային զանգվածը տրոհել մի քանի զանգվածների, դրանց վրա կիրառել տեսակավորման ալգորիթմն առանձին հոսքերով, վերջում դրանք միավորել ելքային զանգվածի մեջ: Այս մեթոդի դեպքում նոր ալգորիթմի բարդությունը կանխատեսելի է: Այն կարելի է ներկայացնել որպես տրոհման, տեսակավորման, միավորման բարդությունների գումար: Այս մոտեցումը, իհարկե, չի կարող արդյունավետ լինել բոլոր ալգորիթմների դեպքում, սակայն այն շատ լավ է աշխատում պղպջակային տեսակավորման ալգորիթմի դեպքում: Վերջինիս արագացման փորձարարական գրաֆիկը, կախված հոսքերի քանակից, պատկերված է նկ. 1-ում:



Նկ. 1. Պղպջակային ալգորիթմի զուգահեռացումը հոսքերի միջոցով

Նկ.1-ում պատկերված է արագացման գրաֆիկը 1,000, 10,000 և 100,000 տարրեր պարունակող զանգվածների դեպքում: Ինչպես երևում է գրաֆիկից՝ այս մեթոդն ավելի արդյունավետ է մեծ զանգվածների դեպքում: Որպես մեթոդի թերություն կարելի է նշել այն, որ անհրաժեշտ է լրացուցիչ հիշողություն՝ մուտքային զանգվածի հիշողության չափով:

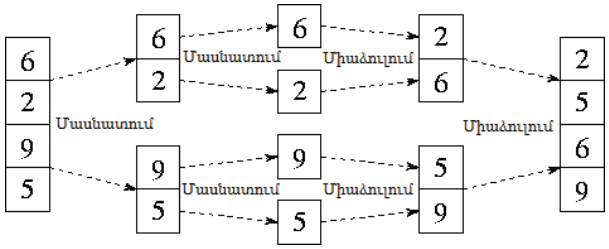
Հարկ է նշել այն, որ հոսքերի որոշակի քանակից սկսած արագացումը սկսում է նվազել: Կոնկրետ մեր փորձարարական արդյունքում հոսքերի այդ թիվը մոտավորապես քսանչորսն է:

Զուգահեռացում ալգորիթմի փոփոխության միջոցով: Ալգորիթմի փոփոխության միջոցով զուգահեռացումը հատուկ է տվյալ ալգորիթմին: Դիտարկենք պղպջակային, միաձուլման և արագ ալգորիթմերի զուգահեռացման տարբերակներ: Որպեսզի երկու տարբեր ծրագրային հրամաններ կարողանան կատարվել զուգահեռ, անհրաժեշտ է, որ դրանք բավարարեն Բերնստայնի պայմանները [3]: Ենթադրենք՝ P_i և P_j-ն ծրագրային հրամաններ են: P_i-ի համար I_i-ն բոլոր մուտքային, իսկ, O_i-ն բոլոր ելքային փոփոխականներն են, նույն կերպ՝ նաև P_j-ի դեպքում: Կարելի է ասել, որ P_i-ն և P_j-ն իրարից անկախ են, եթե՝

$$I_j \cap O_i = \varnothing,\ I_i \cap O_j = \varnothing,\ O_i \cap O_j = \varnothing$$

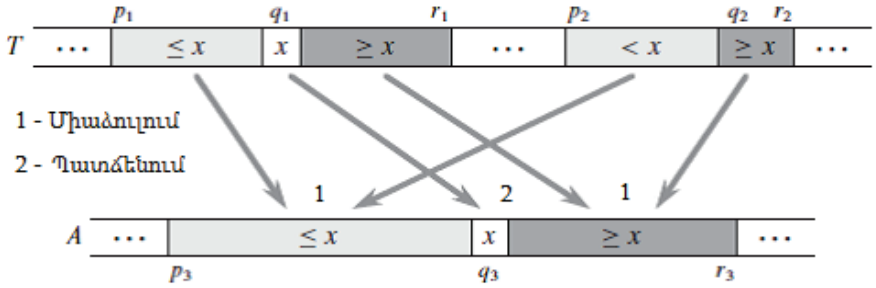
Փաստորեն ալգորիթմը կարելի է փոխակերպել այնպես, որ հաճախակի կրկնվող գործողություններին համապատասխանող հրամանները բավարարեն վերը նշված պայմանները: Այդ դեպքում զուգահեռացում կարելի է ստանալ կոնվեյերի մեխանիզմի միջոցով: Պղպջակային ալգորիթմի դեպքում ցիկլի յուրաքանչյուր իտերացիայի հրամանները կախված են նախորդից: Այդ կախվածությունը վերացնելու համար կարելի է օգտվել կենտ-զույգ տրանսպոնացման մեթոդից: Այս դեպքում մուտքային զանգվածը պետք է բաժանել երկու պայմանական մասերի՝ կենտերի և զույգերի: Առաջին անցման դեպքում կհամեմատվի կենտ ինդեքսով տարրն իր աջ հարևան զույգ ինդեքսով տարրի հետ, հաջորդ անցմանը՝ զույգ ինդեքսով տարրն իր աջ հարևան կենտ ինդեքսով տարրի հետ: Կենտ և զույգ անցումները կրկնվում են այնքան ժամանակ, մինչև ոչ մի փոփոխություն չկատարվի [4]: Զուգահեռացման այս եղանակի դեպքում մուտքային զանգվածը հնարավոր է տեսակավորել n անցումներով, որոնցից յուրաքանչյուրը կպահանջի $n/2$ համեմատում-փոխանակում գործողություն: Ճիշտ է, ալգորիթմի բարդությունը մնում է քառակուսային, սակայն այս մեթոդը փորձնականորեն տալիս է 1.8-2.3 անգամ արագացում՝ անկախ մուտքային զանգվածի տարրերի քանակից: Այս մեթոդը չի պահանջում հավելյալ հիշողություն: Ուշագրավ է այս և նախորդ (բազմահոսքային) մեթոդների համադրումը, որի արդյունքում փորձնականորեն ստացված արագացումը լինում է մոտ երկու անգամ ավելի մեծ, քան ուղղակի բազմահոսքային դեպքում:

Միաձուլման ալգորիթմը պատկանում է «բաժանիր և տիրիր» (անգլերեն՝ divide and conquer) ալգորիթմների դասին և ունի ռեկուրսիվ բնույթ, ինչի շնորհիվ էլ կարող է լավ զուգահեռացվել: Սխեմատիկորեն միաձուլման ալգորիթմը պատկերված է նկ. 2-ում՝



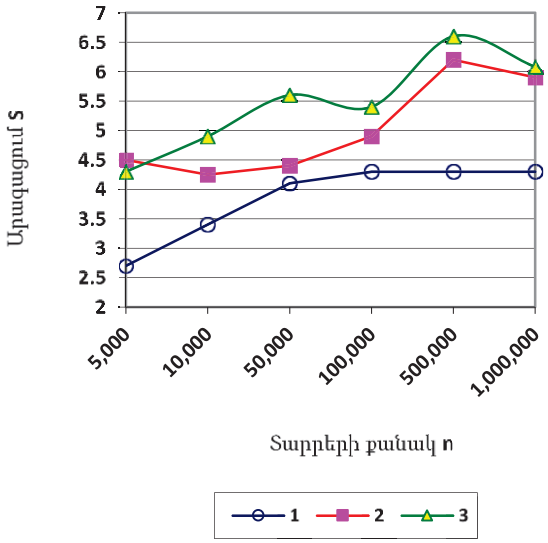
Նկ. 2. Միաձուլման տեսակավորման ալգորիթմի սխեմատիկական ներկայացումը

Զուգահեռացման ամենապարզ ձևը կլինի այն, որ յուրաքանչյուր ռեկուրսիվ կանչ կատարվի առանձին հոսքով: Բայց փորձը ցույց է տալիս, որ այս մեթոդն առանձնապես չի բարձրացնում արագագործությունը: Եթե հաջորդական ալգորիթմը պահանջում է $log(n)$ անգամ կատարել n գործողություն, ապա այստեղ այդ գործողությունները կատարվում են զուգահեռ, և պահանջված ժամանակը կլինի n բարդության, այսպիսով՝ արագացումը կլինի $log(n)$, որն իրականում փոքր թիվ է [2]: Այստեղից կարելի է եզրակացնել, որ այս ալգորիթմում հիմնական ժամանակատար մասը վերջին՝ միաձուլման պրոցեսն է, որը կատարվում է հաջորդաբար: Միաձուլումը կարող է զուգահեռացվել, ինչպես ցույց է տրվում [2]-ում: Սխեմատիկորեն այն ներկայացված է նկ. 3-ում: Չնայած այս ձևով հասնում ենք $O(n/log(n^2))$ արագացման, այնուամենայնիվ, սա գործնականում չի իրագործվում: Այս ձևով ալգորիթմը ստեղծում է մեծ քանակությամբ հոսքեր, ինչն էլ հանգեցնում է հիշողության և պրոցեսորի ռեսուրսների վատնման: Այս խնդրից խուսափելու համար կարելի է դնել որոշակի սահմանափակումներ ռեկուրսիայի խորության վրա: Դա կարելի է իրականացնել երկու ճանապարհով: Նախ սահմանել տարրերի քանակի որոշակի շեմ, որից սկսած ալգորիթմը կսկսի աշխատել հաջորդաբար՝ առանց նոր հոսքերի օգտագործմամբ: Բայց այս եղանակը չի կարող լիովին լուծել խնդիրը: Մուտքային զանգվածի չափը փոփոխական է, իսկ շեմը պետք է լինի նախօրոք սահմանված, ուստի մուտքային զանգվածի բավականին մեծ լինելու դեպքում նույնպես հոսքերի քանակը կարող է շատ լինել:



Նկ. 3. Զուգահեռ միաձուլում

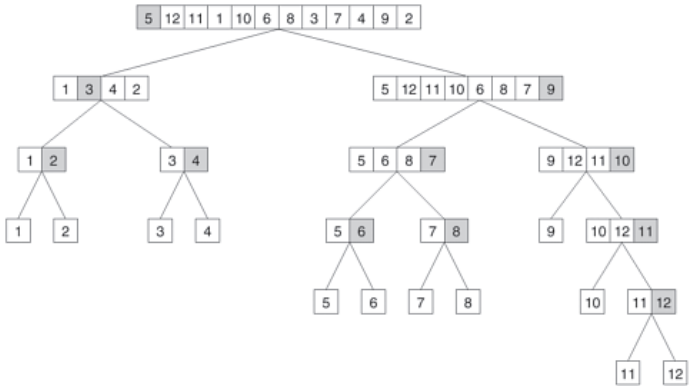
Դա վերացնելու համար պարզապես կարելի է սահմանել ռեկուրսիայի խորությունը և դադարեցնել նոր հոսքերի ստեղծումը այդ խորությանը հասնելու դեպքում: Տվյալ խորությունը նպատակահարմար է ընտրել՝ համակարգչի պրոցեսորի միջուկների քանակից ելնելով: Փորձարարական արդյունքը ներկայացված է նկ. 4-ում:



Նկ. 4. Զուգահեռացված միաձուլման տեսակավորման ալգորիթմի արագացումը

Նկ. 4-ում ցույց է տրվում արագացումը 1, 2, 3 խորությունների դեպքում, իսկ որպես հաջորդական տեսակավորման շեմ ընտրված է 1024: 4 խորության դեպքում գրաֆիկը ստացվում է բավականին նման 3 խորության դեպքում ստացվող գրաֆիկին, իսկ 5-ի դեպքում արագացումը սկսում է նվազել: Նկ. 4-ից երևում է, որ ալգորիթմն ավելի արդյունավետ է մեծ զանգվածների դեպքում: Ալգորիթմը նաև պահանջում է մուտքային զանգվածի հիշողության չափով լրացուցիչ հիշողություն:

Սկզբունքորեն նույն մոտեցումը կարելի է ցուցաբերել նաև արագ տեսակավորման ալգորիթմի զուգահեռացման դեպքում, որը նույնպես պատկանում է «բաժանիր և տիրիր ալգորիթմների» դասին և ունի ռեկուրսիվ բնույթ: Արագ ալգորիթմը սխեմատիկորեն պատկերված է նկ. 5-ում: Սակայն այս դեպքում ստացված արագացումը, համեմատած միաձուլման ալգորիթմի զուգահեռացման դեպքում ստացված արագացմանը, այդքան էլ մեծ չէ ($S = 1 \div 2$):



Նկ. 5. Արագ տեսակավորման ալգորիթմի սխեմատիկ ներկայացում

Այդ պահվածքը բացատրվում է նրանով, որ ի տարբերություն միաձուլման տեսակավորման ալգորիթմին՝ արագ տեսակավորման ալգորիթմի դեպքում յուրաքանչյուր անգամ ռեկուրսիվ կանչն իրականացվում է ոչ թե ընթացիկ զանգվածի կեսի վրա, այլ նրա ինչ-որ մի մասի: Վերջինիս չափը որոշվում է հենակետային տարրի ընտրությունից, որն էլ երաշխավորված չէ, որ կլինի հենց միջին արժեքով տարրը: Դա հանգեցնում է հոսքերի միջև աշխատանքի ոչ հավասարաչափ բաշխմանը: Այդ խնդիրը կարելի է մեղմացնել հենակետային տարրի փնտրման ալգորիթմի կատարելագործմամբ: Իհարկե, հնարավոր է ընտրել հենակետային տարրն այնպես, որ այն լինի տարրերից միջին արժեք ունեցողը, սակայն դա պահանջում է լրացուցիչ գործողություններ, որոնք իրենց հերթին կդանդաղեցնեն ալգորիթմը: Այդ պատճառով էլ գործնականում նպատակահարմար է որպես հենակետային տարր ընտրել որևէ պատահական տարր:

Եզրակացություն: Այսպիսով, աշխատանքում ներկայացված տեսակավորման ալգորիթմների զուգահեռացման մի քանի մեթոդներն արդյունավետ կարող են լինել տարբեր կամ կոնկրետ մեկ ալգորիթմի դեպքում: Նոր ալգորիթմների օգտագործմամբ, փորձարարական ճանապարհով, պղպջակային տեսակավորման ալգորիթմն արագացել է ընդհուպ մինչև 140 անգամ, միաձուլման ալգորիթմը՝ 6.5, արագ ալգորիթմը՝ 2: Որոշ դեպքերում նոր ալգորիթմները պահանջում են հավելյալ ռեսուրսներ (ինչպես, օրինակ՝ օպերատիվ հիշողություն), սակայն ժամանակային կրիտիկական խնդիրների լուծման համար դրանք կարող են շատ ավելի նպատակահարմար լինել, քան իրենց նախնական տարբերակները:

ԳՐԱԿԱՆՈՒԹՅԱՆ ՑԱՆԿ

1. **Pacheco P. S.** An Introduction to parallel programming. - University of San Francisco, 2011.
2. **Cormen T. H., Leiserson C. E., Rivest R. L., Stein C.** Introduction to algorithms. 3rd ed.- Cambridge, Massachusetts, London, England, The MIT Press, 2009.
3. **Bernstein A. J.** Analysis of programs for parallel //Processing IEEE Trans. Electronic Computers. - Oct. 1966. - Vol. 15. - P. 757-763.
4. **Breshears C.** The art of concurrency. - O'Reilly, 2009.

Н.О. АБРОЯН, Р.Г. АКОБЯН

ИССЛЕДОВАНИЕ И РАЗРАБОТКА МЕТОДОВ ПАРАЛЛЕЛИЗАЦИИ АЛГОРИТМОВ СОРТИРОВКИ

Произведены исследование, разработка и эксперименты нескольких методов параллелизации разных алгоритмов сортировки. Показано, как можно с помощью метода параллелизации повысить производительность алгоритма (как, например, методы пузырчатой сортировки, сортировки слиянием, быстрой сортировки) и сравнить с его последовательной версией.

Ключевые слова: алгоритмы сортировки, пузырчатая сортировка, сортировка слиянием, быстрая сортировка, потоки, ускорение.

N.H. ABROYAN, R.G. HAKOBYAN

RESEARCH AND ELABORATION OF PARALLELIZATION METHODS FOR SORTING ALGORITHMS

Research, elaboration and experiments of several parallelization methods are performed for different sorting algorithms. It is shown how the performance of an algorithm (such as bubble sort, merge sort, quick sort algorithms) can be improved through parallelization and compared with its sequential version.

Keywords: sorting algorithms, bubble sort, merge sort, quick sort, threads, speedup.