Fakulta riadenia a informatiky
Informatika

Domáce zadanie 1

Analýza výkonu ADT dvojrozmerné pole

doc. Ing. **Miroslav Kvaššay** PhD.
STREDA 12, 13 Maroš Gorný, 5ZYI21
2021/2022

FRI UNIZA

Obsah

Analýza výkonu ADT dvojrozmerné pole 1
Úloha 2 3
Zadanie – Overenie výkonu v scenári 3
Testovanie – Implementácia 3
Testovanie – Výsledky 5
Úloha 3 13
Zadanie – analýza časových zložitostí 13
Testovanie – Implementácia 13
Testovanie – Výsledky 14

Úloha 2

Zadanie – Overenie výkonu v scenári

Úloha 2 – overenie výkonu v scenári
Realizácie ADT dvojrozmerné pole – matica, ktoré ste implementovali v úlohe 1, otestujte v scenároch definovaných v Tab. 1. V každom scenári vykonajte spolu **1 000 000** operácií. Jednotlivé operácie sú v jednotlivých scenároch volané náhodne tak, aby na konci súhlasil podiel jednotlivých operácií. Parametre do operácií sú taktiež náhodné, pričom ako index riadku a index stĺpca je nutné náhodne zvoliť akýkoľvek aktuálne platný index.

Tab. 1 Testovacie scenáre pre ADT dvojrozmerné pole – matica

Scenár	**Rozmery matice**	**Podiel operácií**		
		`getRowCount`	`getColumnCount`	`at`
A	10 x 50	5	5	90
B	2000 x 500	5	5	90
C	50 x 10	10	30	60
D	500 x 2000	10	30	60

V rámci analýzy výkonu v scenári je nutné merať len dĺžku trvania vybranej operácie. To znamená, že do merania **sa nesmie započítavať čas potrebný pre generovanie pomocných údajov**.

Testovanie – Implementácia

Použité inštancie a funkcie

Matrix<T>* - Matica s ktorou budem pracovať.
SimpleTest - Inštancia s ktorou môžem merať čas
FileLogConsumer - Logger s ktorým budem schopný ukladať dáta.
srand(time(NULL)); - Funkcia s ktorou môžem nastaviť náhodný seed pre funkciu rand().
RAND_MAX - Konštanta definovaná v <cstdlib>.
rand() - Funkcia ktorá vráti náhodné číslo medzi 0 a RAND_MAX.

Logika testovania

Výber vhodnej implementácie

Konečný podiel operácií má byť rovný podielu zvoleného v tabuľke. Jedna z možností ako to spraviť je, že by sme najprv pridali podiel prvej operácie, potom druhej a nakoniec poslednej.

Avšak príde mi vhodnejšie dané operácie volať s pravdepodobnosťou takou, akou majú pomer. Preto budem používať funkciu „rand() % 100” ktorá mi vygeneruje čísla od 0 po 99 a následne podľa pravdepodobnosti volať dané operácie.

Mám preto vypočítané tri hranice, vďaka ktorým sa môžem rozhodovať ktorú operáciu zavolám keď sa vygeneruje náhodné číslo od 0 po 99. Vždy volám operáciu podľa toho, či sa náhodné číslo nachádza pod prvou hranicou, ak nie, spýtam sa to isté ďalšej hranice. Spolu musím vykonať 1 000 000 operácií.

1. Hranica – Pomer 1. operácie (Ak má 1. operácie pomer 5, 1 hranica sa bude rovnať tiež 5)
    a. Volanie metódy getRowCount().
2. Hranica – Pomer 1. operácie + pomer 2. operácie
    a. Volanie metódy getColumnCount().
3. Hranica – 100
    a. Volanie metódy at().

Týmto spôsobom sa priblížim k tomu, že dané operácie sa budú vykonávať podľa daného pomeru.

Môže sa ale stať, že daná operácia sa vykoná už maximálny počet krát, ale pravdepodobnosť mi určí, že by sa mala vykonať ešte raz. Keďže chcem dodržať pomer, musím v takomto prípade vybrať inú operáciu ktorá ešte nedosiahla maximálny počet vykonania. Výber ďalšej operácie v prípade, že predošlá operácia dosiahla svoje maximum, robím tak, že jednoducho si vyberiem prvú voľnu operáciu ktorá ešte nedosiahla maximum. To však môže mať za následok to, že na konci sa mi jedna operácia bude opakovať viac krát za sebou.

Ukladanie údajov

Dáta som ukladal do CSV súboru, kde

1. Stĺpec – počet riadkov matice
2. Stĺpec – počet stĺpcov matice
3. Stĺpec – čas metódy v nanosekundách
4. Stĺpec – volaná metóda

Rows	Columns	Time[ns]	Method
10	50	127800	at
10	50	1800	getColumn
10	50	1300	at
10	50	400	at
10	50	400	at
10	50	300	at
10	50	400	at
10	50	1200	getRowCou

Table 1 - Ukážka CSV súboru pre úlohu 1

Testovanie – Výsledky

Na vyhodnotenie daných výsledkov som použil kontingenčnú tabuľku, kde na X osi je priemer danej metódy a na osi Y je čas v nanosekundách [ns].

Taktiež som si spravil pomocnú tabuľku ku každej dvojici v danom scenári.

	ABS Diff: Coh vs Incoh				
	at	getColumn	getRow	avgGet	at vs avgGet
Diff	75,4123333	43,166	29,64		
Coh	403,142333	387,716	395,124	391,42	11,72233333
Incoh	478,554667	430,882	424,764	427,823	50,73166667

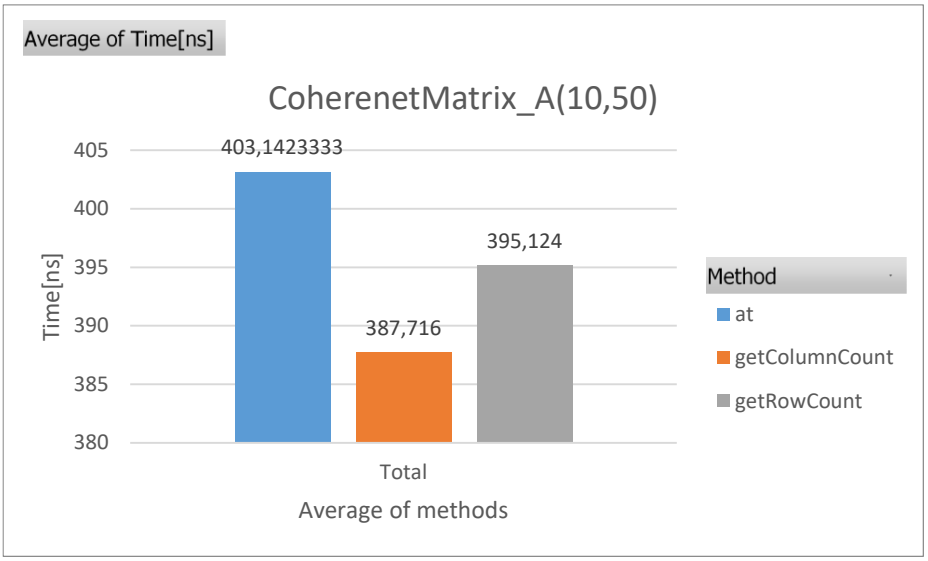
Kde v prvom riadku mám absolútny rozdiel času medzi maticou A a B, v druhom a treťom riadku je čas daných metód pre danú maticu.
avgGet je priemer metód getColumnCount a getRowCount a at vs avgGet je absolútny rozdiel týchto dvoch metód oproti metóde at.

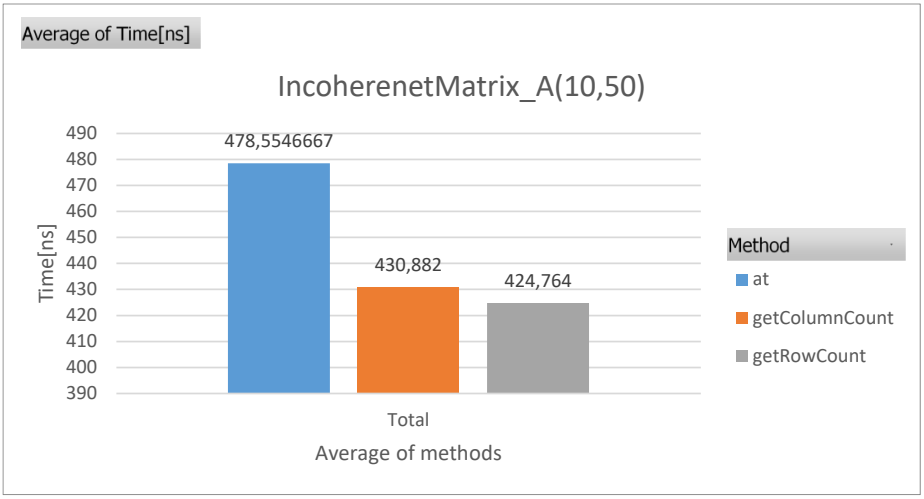
Scenár A

Zadanie

Scenár	**Rozmery matice**	**Podiel operácií**		
		`getRowCount`	`getColumnCount`	`at`
A	10 x 50	5	5	90

Prezentácia výsledkov





	ABS Diff: Coh vs Incoh				
	at	getColumn	getRow	avgGet	at vs avgGet
Diff	75,4123333	43,166	29,64		
Coh	403,142333	387,716	395,124	391,42	11,72233333
Incoh	478,554667	430,882	424,764	427,823	50,73166667

Vyhodnotenie výsledkov

Na základe grafov môžeme vidieť, že

Coherent Matrix

- at – Bola v priemere pomalšia o 12 nanosekúnd ako ostatné metódy.
- getColumn, getRow – O niečo rýchlejšie ako metóda at, avšak len o pár nanosekúnd, ktoré nie sú veľmi významné.

Incoherent Matrix

- at – Bola v priemere pomalšia o 51 nanosekúnd ako ostatné metódy.
- getColumn, getRow – Tu boli obidve metódy takmer tak isto rýchle a vidíme, že boli rýchlejšie ako metóda at.

Coherent vs Incoherent

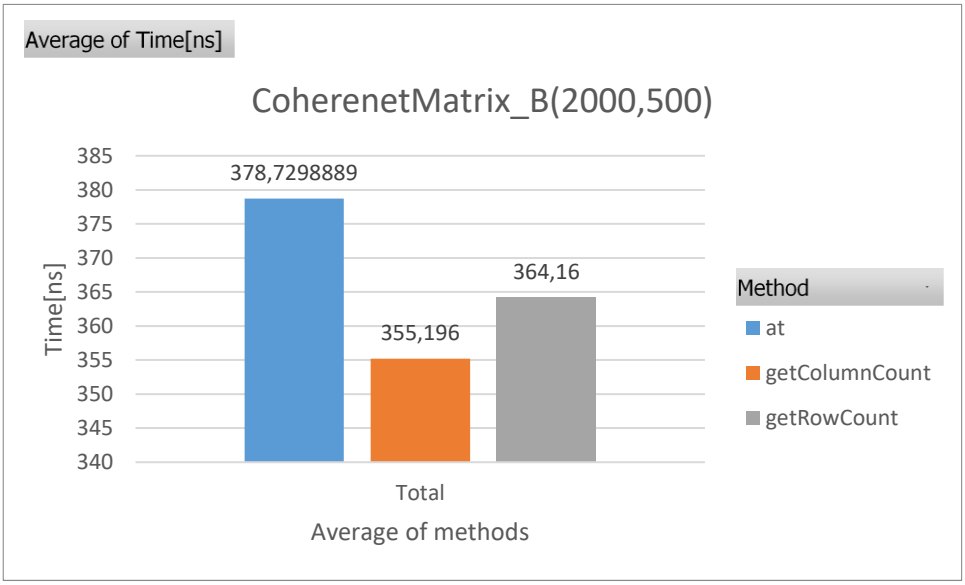
- Vidíme, že dané metódy matice so **súvislou pamäťou** sú pri danom počte prvkov a riadkov **rýchlejšie** ako dané metódy matice s nesúvislou pamäťou.
- Metóda at je horšia ako ostatné metódy, avšak pri danom počte riadkov a stĺpcov je to stále zanedbateľné množstvo. Ale vidíme, že v matici so **súvislou pamäťou** je metóda at **rýchlejšia** o 75 nanosekúnd ako v matici s nesúvislou pamäťou.

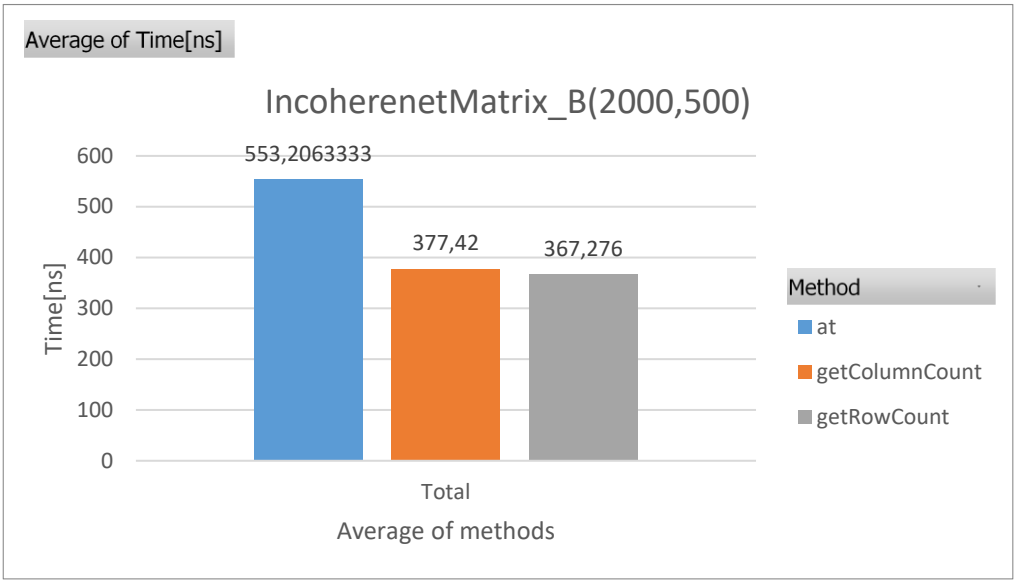
Scenár B

Zadanie

Scenár	**Rozmery matice**	**Podiel operácií**		
		`getRowCount`	`getColumnCount`	`at`
B	2000 x 500	5	5	90

Prezentácia výsledkov





	ABS Diff: Coh vs Incoh				
	at	getColumn	getRow	avgGet	at vs avgGet
Diff	174,476444	22,224	3,116		
Coh	378,729889	355,196	364,16	359,678	19,05188889
Incoh	553,206333	377,42	367,276	372,348	180,8583333

Vyhodnotenie výsledkov

Na základe grafov môžeme vidieť, že

Coherent Matrix

- at – Bola v priemere pomalšia o 19 nanosekúnd ako ostatné metódy.

- getColumn, getRow – O niečo rýchlejšie ako metóda at, avšak len o pár nanosekúnd, ktoré nie sú veľmi významné a metóda getColumn o trošku rýchlejšia ako metóda getRow.

Incoherent Matrix

- at – Bola v priemere pomalšia o 181 nanosekúnd ako ostatné metódy.
- getColumn, getRow – Tu boli obidve metódy takmer tak isto rýchle a vidíme, že boli rýchlejšie ako metóda at.

Coherent vs Incoherent

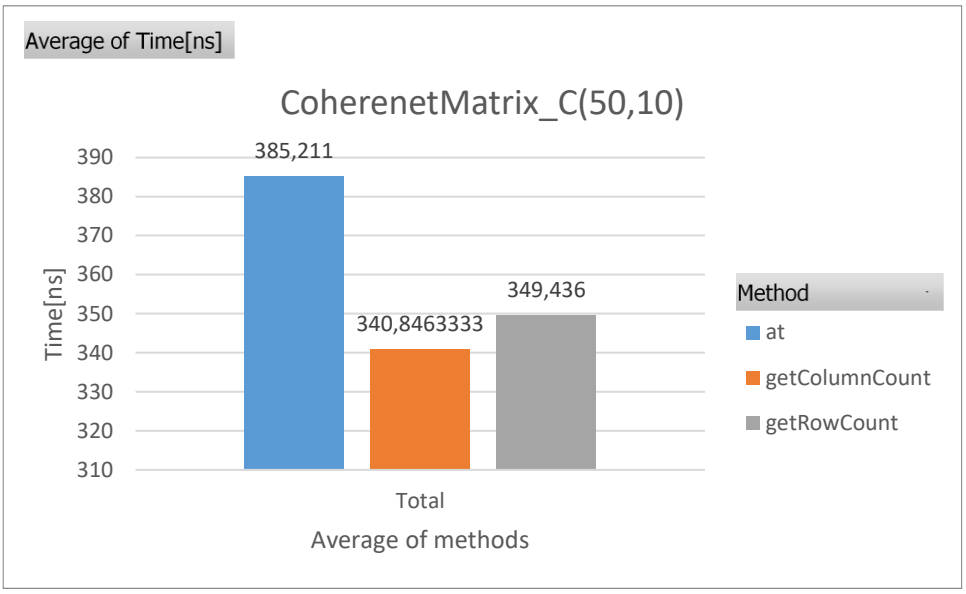
- Vidíme, že metóda at matice so **súvislou pamäťou** je pri danom počte prvkov a riadkov značne **rýchlejšia** o 174 nanosekúnd ako metóda at matice s nesúvislou pamäťou.
- Metódy getColumnCount a getRowCount boli v obidvoch prípadoch podobne rýchle.

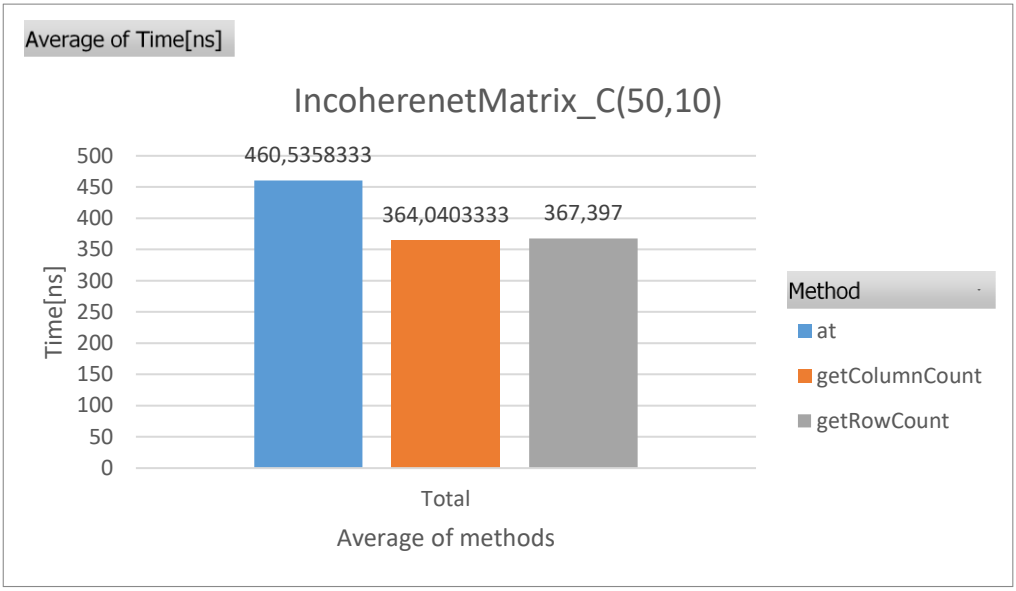
Scenár C

Zadanie

Scenár	Rozmery matice	Podiel operácií		
		`getRowCount`	`getColumnCount`	`at`
C	50 x 10	10	30	60

Prezentácia výsledkov





	ABS Diff: Coh vs Incoh				
	at	getColumn	getRow	avgGet	at vs avgGet
Diff	75,3248333	23,194	17,961		
Coh	385,211	340,846333	349,436	345,141167	40,06983333
Incoh	460,535833	364,040333	367,397	365,718667	94,81716667

Vyhodnotenie výsledkov

Na základe grafov môžeme vidieť, že

Coherent Matrix

- at – Bola v priemere pomalšia o 40 nanosekúnd ako ostatné metódy.
- getColumn, getRow – Tu boli obidve metódy takmer tak isto rýchle a vidíme, že boli rýchlejšie ako metóda at.

Incoherent Matrix

- at – Bola v priemere pomalšia o 94 nanosekúnd ako ostatné metódy.
- getColumn, getRow – Tu boli obidve metódy takmer tak isto rýchle a vidíme, že boli rýchlejšie ako metóda at.

Coherent vs Incoherent

- Vidíme, že metóda at matice so **súvislou pamäťou** je pri danom počte prvkov a riadkov **rýchlejšia** o 75 nanosekúnd ako metóda at matice s nesúvislou pamäťou.
- Metódy getColumnCount a getRowCount boli v obidvoch prípadoch podobne rýchle.

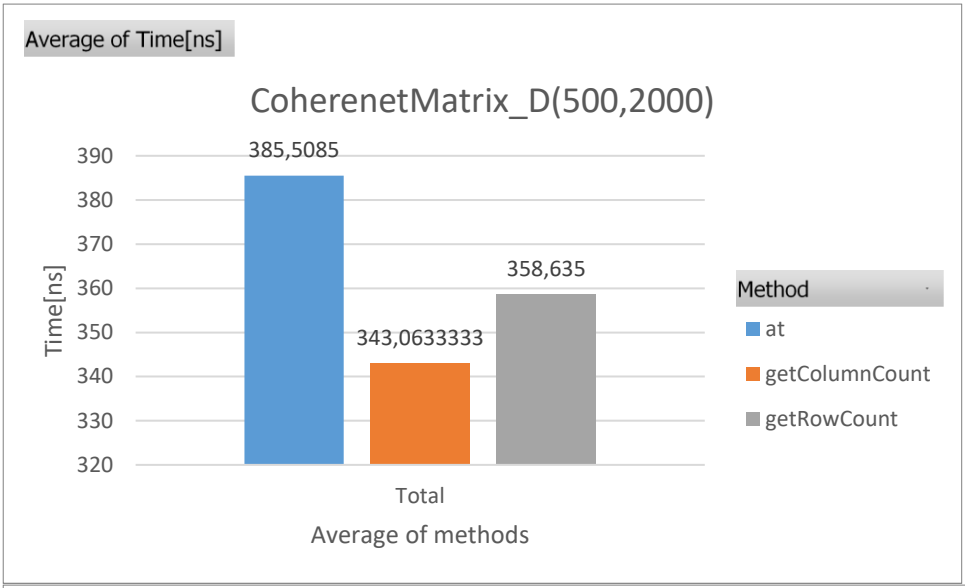
Scenár D

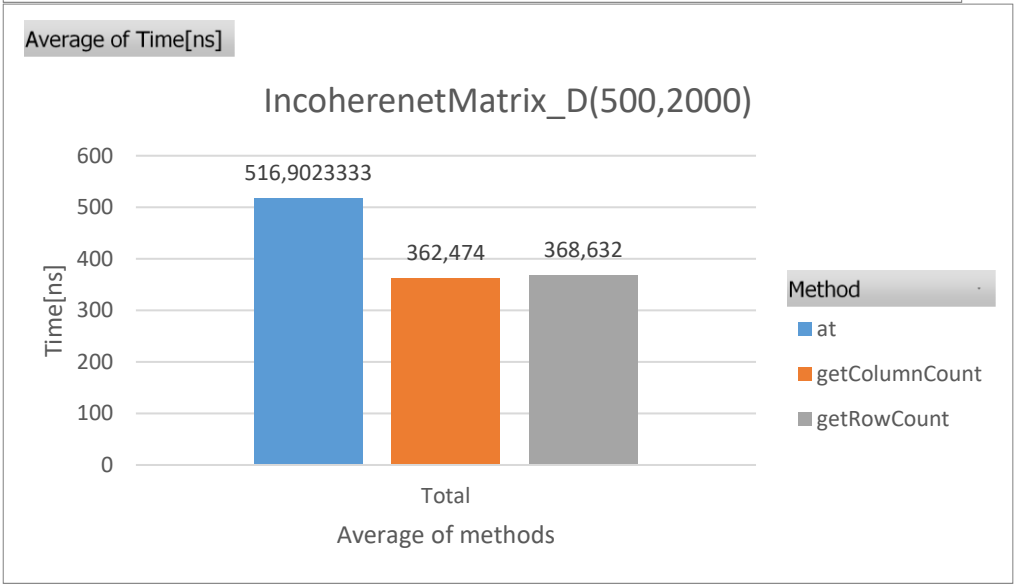
Zadanie

Scenár	Rozmery matice	Podiel operácií

		getRowCount	getColumnCount	at
D	500 x 2000	10	30	60

Prezentácia výsledkov





	ABS Diff: Coh vs Incoh				
	at	getColumn	getRow	avgGet	at vs avgGet
Diff	131,393833	19,4106667	9,997		
Coh	385,5085	343,063333	358,635	350,849167	34,65933333
Incoh	516,902333	362,474	368,632	365,553	151,3493333

Vyhodnotenie výsledkov

Na základe grafov môžeme vidieť, že

Coherent Matrix

- at – Bola v priemere pomalšia o 34 nanosekúnd ako ostatné metódy.

- getColumn, getRow – O niečo rýchlejšie ako metóda at, avšak len o pár nanosekúnd, ktoré nie sú veľmi významné a metóda getColumn o trošku rýchlejšia ako metóda getRow.

Incoherent Matrix

- at – Bola v priemere pomalšia o 151 nanosekúnd ako ostatné metódy.
- getColumn, getRow – Tu boli obidve metódy takmer tak isto rýchle a vidíme, že boli rýchlejšie ako metóda at.

Coherent vs Incoherent

- Vidíme, že metóda at matice so **súvislou pamäťou** je pri danom počte prvkov a riadkov **rýchlejšia** o 131 nanosekúnd ako metóda at matice s nesúvislou pamäťou.
- Metódy getColumnCount a getRowCount boli v obidvoch prípadoch podobne rýchle.

Záver

B vs D

Vďaka výsledkom môžeme zhodnotiť, že keď porovnáme matice s väčším počtom riadkov a stĺpcov aké boli v scenári B a D, tak jasne vidíme,
že značne rýchlejšia je matica so súvislou pamäťou. Pričom pri väčšom počte riadkov ako stĺpcov mala matica s nesúvislou pamäťou väčší problém a matica so súvislou pamäťou sa správala tak isto rýchlo bez ohľadu na to, či mala viac riadkov alebo stĺpcov.

	B(2000,500)_ABS Diff: Coh vs Incoh				
	at	getColumn	getRow	avgGet	at vs avgGet
Diff	174,476444	22,224	3,116		
Coh	378,729889	355,196	364,16	359,678	19,05188889
Incoh	553,206333	377,42	367,276	372,348	180,8583333

	D(500,2000)_ABS Diff: Coh vs Incoh				
	at	getColumn	getRow	avgGet	at vs avgGet
Diff	131,393833	19,4106667	9,997		
Coh	385,5085	343,063333	358,635	350,849167	34,65933333
Incoh	516,902333	362,474	368,632	365,553	151,3493333

A vs C

Pri porovnaní matíc s menším počtom riadkov a stĺpcov aké boli v scenári A a C, tak vidíme,
že znova bola rýchlejšia matica so súvislou pamäťou avšak prekvapivo sú metódy getRow a getColumn rýchlejšie v matici ktorá mala menej stĺpcov ako riadkov.

	A(10,50)_ABS Diff: Coh vs Incoh				
	at	getColumn	getRow	avgGet	at vs avgGet
Diff	75,4123333	43,166	29,64		
Coh	403,142333	387,716	395,124	391,42	11,72233333
Incoh	478,554667	430,882	424,764	427,823	50,73166667
	C(50,10)_ABS Diff: Coh vs Incoh				
	at	getColumn	getRow	avgGet	at vs avgGet
Diff	75,3248333	23,194	17,961		
Coh	385,211	340,846333	349,436	345,141167	40,06983333
Incoh	460,535833	364,040333	367,397	365,718667	94,81716667

Všetky scenáre

Môžeme povedať, že matica so súvislou pamäťou je rýchlejšia po všetkých stránkach. Matica s nesúvislou pamäťou sa s narastajúcou veľkosťou spomaľuje a pri väčšom počte riadkov ako stĺpcov sa spomaľuje viac ako keď sa jej zväčšujú len stĺpce.

Úloha 3

Zadanie – analýza časových zložitostí

Úloha 3 – analýza časových zložitostí

V rámci analýzy časových zložitostí je nutné otestovať rýchlosť operácií `at` a `assign` v závislosti od rozmerov (*m* – počet riadkov, *n* – počet stĺpcov) a implementácie dvojrozmerného poľa – matice. Za týmto účelom je nutné meniť hodnoty parametrov *m* a *n* tak, aby ste zistili, ktorý z týchto parametrov má väčší vplyv na predmetnú operáciu, resp., či hodnoty týchto parametrov ovplyvňujú rýchlosť uvedenej operácie. Hodnotu každého z parametrov *m* a *n* definujte aspoň z rozsahu **1 .. 1 000** (napr. s krokom 100), pričom pri každej vygenerovanej matici vykonajte aspoň 100-krát testovanú operáciu a následne vypočítajte priemerný čas potrebný pre beh testovanej operácie pri daných rozmeroch matice.

a) Pre realizácie ADT dvojrozmerné pole – matica, ktoré ste implementovali v úlohe 1, odhadnite zložitosť operácie `at`. Parametre testovanej operácie sú náhodné, pričomako index riadku a index stĺpca je nutné náhodne zvoliť akýkoľvek aktuálne platný index. Na základe vykonaných experimentov **odhadnite hornú asymptotickú zložitosť** operácie `at`, pričom **zdôvodnite**, ktorá z dvoch testovaných implementácií ADT dvojrozmerné pole – matica je rýchlejšia.

b) Pre realizácie ADT dvojrozmerné pole – matica, ktoré ste implementovali v úlohe 1, odhadnite zložitosť operácie `assign`. Parametrom testovanej operácie je druhá matica,ktorá je implementovaná rovnako ako testovaná matica a má aj rovnaké rozmery ako testovaná matica. Na základe vykonaných experimentov **odhadnite hornú asymptotickú zložitosť** operácie `assign`, pričom zdôvodnite, ktorá z dvoch testovaných implementácií ADT dvojrozmerné pole – matica je rýchlejšia.

V rámci analýzy časových zložitostí vybraných operácií je nutné merať len dĺžku trvania vybranej operácie. To znamená, že do merania **sa nesmie započítavať čas potrebný pre generovanie pomocných údajov**.

Testovanie – Implementácia

Použité inštancie a funkcie

Matrix<T>* - Matica s ktorou budem pracovať.
SimpleTest - Inštancia s ktorou môžem merať čas
FileLogConsumer - Logger s ktorým budem schopný ukladať dáta.
srand(time(NULL)); - Funkcia s ktorou môžem nastaviť náhodný seed pre funkciu rand().
RAND_MAX - Konštanta definovaná v <cstdlib>.
rand() - Funkcia ktorá vráti náhodné číslo medzi 0 a RAND_MAX.

Logika testovania

Pri testovaní metód at a assign vzhľadom na veľkosť matice, pričom chceme pozorovať vplyv počtu riadkov a stĺpcov, musíme použiť dva cykly kde prvý bude prechádzať cez riadky a druhý cez stĺpce. V mojom prípade som si zvolil maximálny počet riadkov a stĺpcov 3001, pričom začínať budeme na 1 a budeme skákať po 100.

Vytvoríme si pointer na maticu so súvislou alebo nesúvislou pamäťou s počtom riadkov a stĺpcov podľa dvoch cyklov nad nami a vo vnútri cyklu spravíme ďalší cyklus v ktorom 100 krát zavolám metódu ktorá bola vopred zvolená.

V prípade metódy at() si pomocou metódy rand() dva krát vypočítam platný index riadku a stĺpca a zavolám metódu at s danými indexami a rýchlosť metódy odmeriam v nanosekundách.

V prípade metódy assign() si vždy vytvorím nový pointer na novú maticu a na mojej pôvodnej matici zavolám metódu assign() do ktorej vložím dereferencovanú novú maticu a čas danej metódy odmeriam v nanosekundách.

Čas metódy si budem počítať kumulatívne a na konci ho vydelím počtom opakovaní, teda v mojom prípade 100.

Vyčistíme si pamäť a uložíme si dáta.

Ukladanie údajov

Dáta som ukladal do CSV súboru, kde

- 1 bunka bola vždy prázdna.
- Následne boli zapísane do každej ďalšej bunky počty stĺpcov, pričom sa začínalo od 1 a končilo pri 3001 a krok bol 100.
- Na nový riadok sa do prvej bunky zapísal počet riadkov(začínam s 1) a do každej ďalšej bunky sa zapísal čas ako dlho trvala daná metóda podľa počtu riadkov a počtu stĺpcov.
- Keď sme sa dostali na posledný stĺpec, zväčšil sa mi počet riadkov o 100 a opakoval som predošlý bod až pokiaľ počet riadkov nedosiahol 3001.

	1	101	2901	3001
1	5131	205	206	208
101	211	211	282	224
2901	360	292	371	373
3001	234	233	958	365

Ukážka kde boli vynechané riadky a stĺpce.

Testovanie – Výsledky

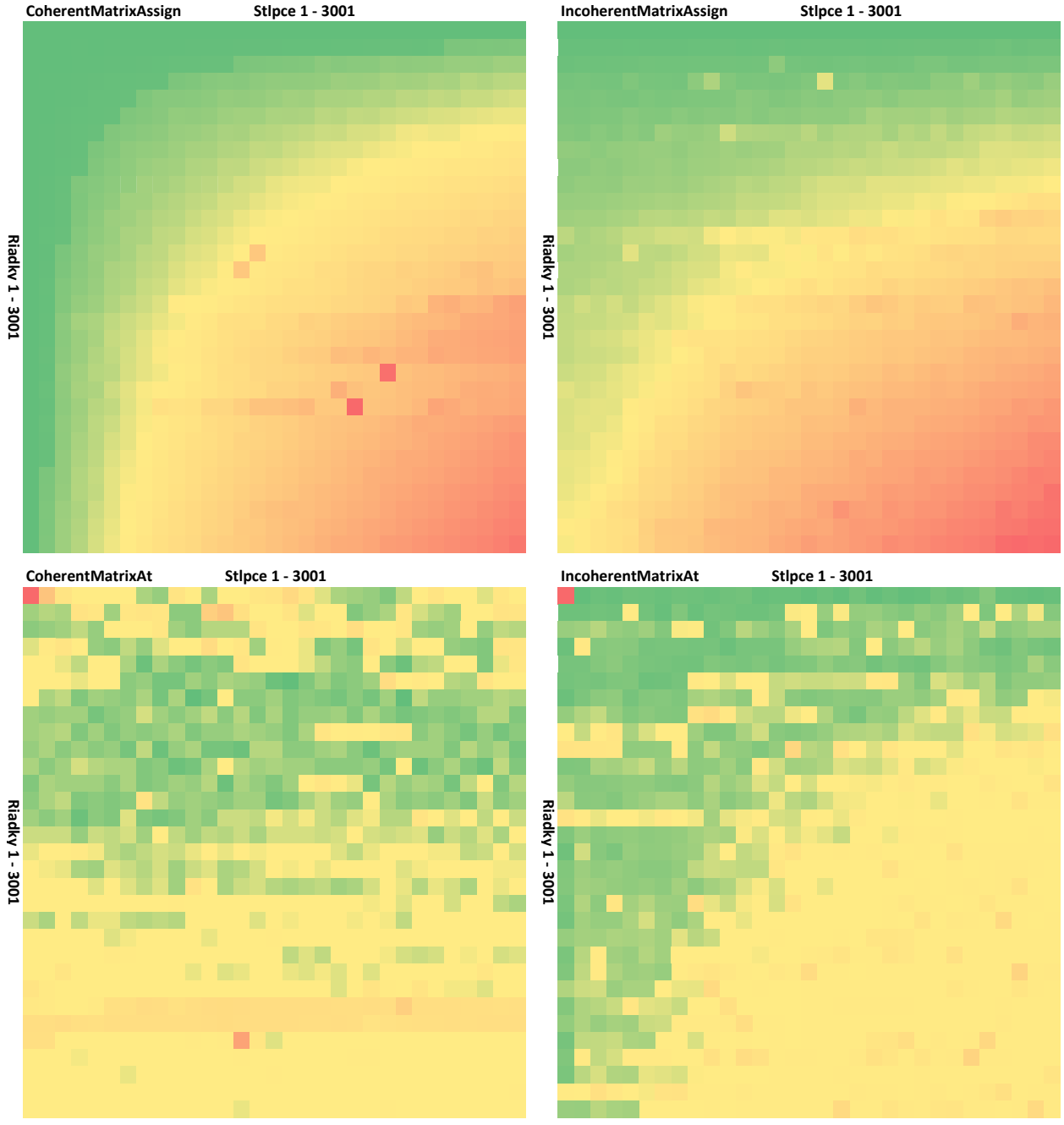
Na vyhodnotenie daných výsledkov som použil heat mapu, kde na X osi sú stĺpce, na Y osi sú riadky a každá bunka predstavuje čas v nanosekundách podľa daného riadku a stĺpca. Keďže mi ide o to, aby som preskúmal vplyv riadkov a stĺpcov na rýchlosť

metód, dané hodnoty som skryl, aby som lepšie mohol vidieť ktorá hodnota je podľa zafarbenia horšia a ktorá lepšia.

Heat mapa – vyhodnotenie vplyvu riadkov a stĺpcov

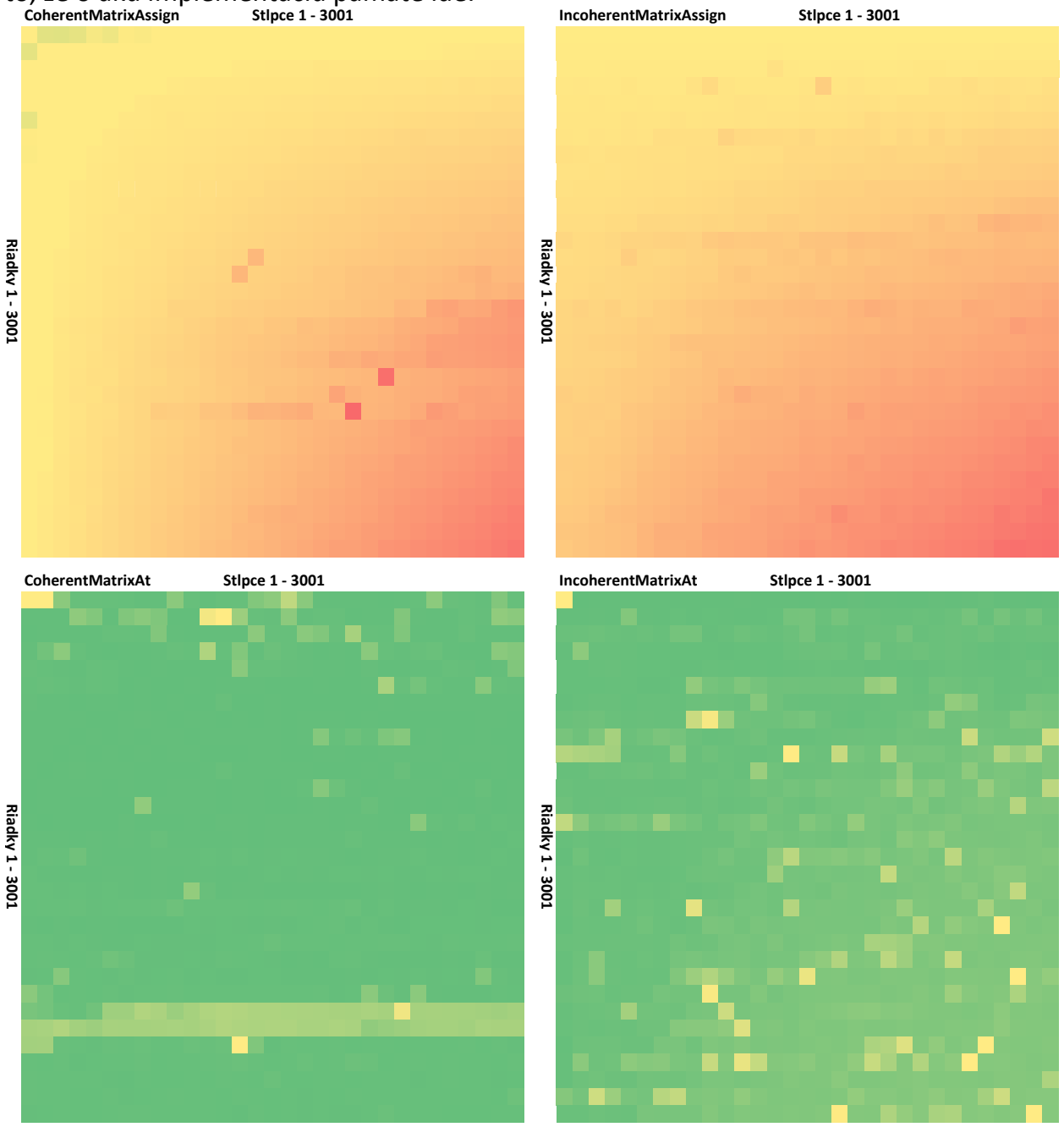
Pre každý scenár zvlášť

Heat mapa je použitá lokálne pre každý scenár zvlášť. Z tohto môžeme vidieť, že pri assign metódach pri súvislej pamäti ma vplyv riadok aj stĺpec rovnako, avšak pri nesúvislej pamäti už má väčší vplyv na rýchlosť riadok. Pri at metódach je tam rozdiel menší, avšak pri súvislej pamäti to vyzerá tak, že na rýchlosť vplýva len počet riadkov pretože riadok je takmer vždy rovnakej farby, takže stĺpec ho neovplyvňuje. Pri nesúvislej pamäti tam je zase menší rozdiel ale tu to vyzerá tak, že vplyv má aj počet riadkov aj počet stĺpcov.



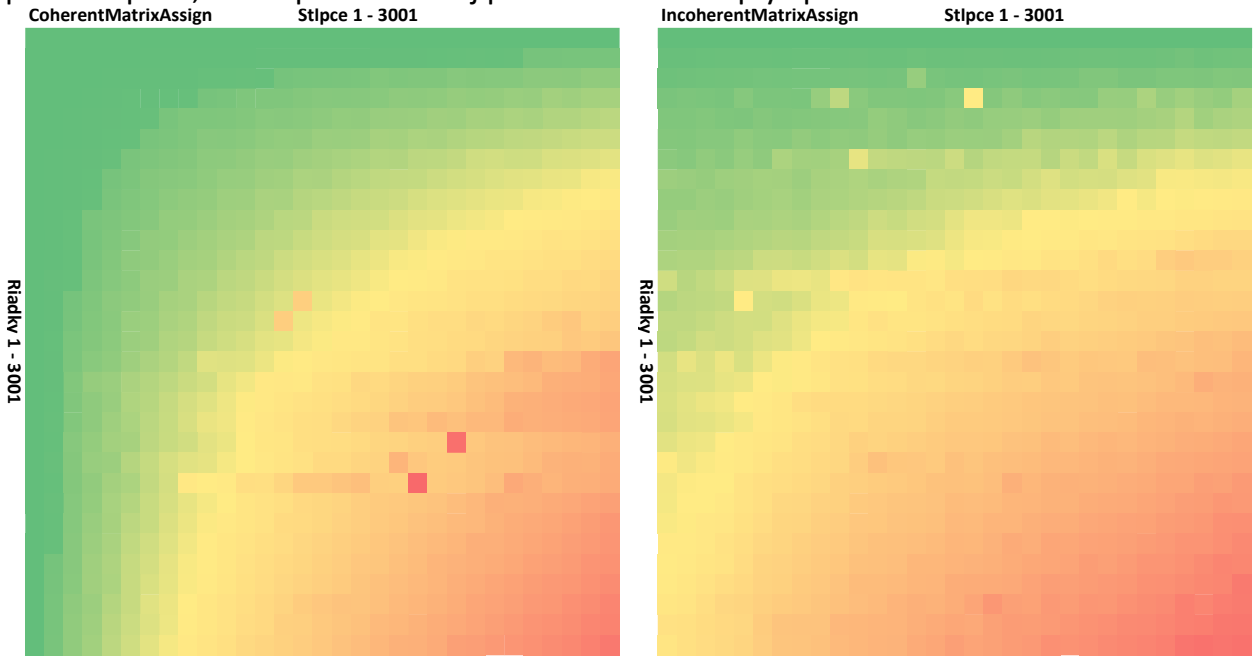
Všetky scenáre spolu

Heat mapa je použitá globálne, takže je vyhodnotená podľa časov zo všetkých scenárov . Z tohto môžeme vidieť, že metóda assign je značne horšie ako metóda at. Bez ohľadu na to, že o akú implementáciu pamäte ide.



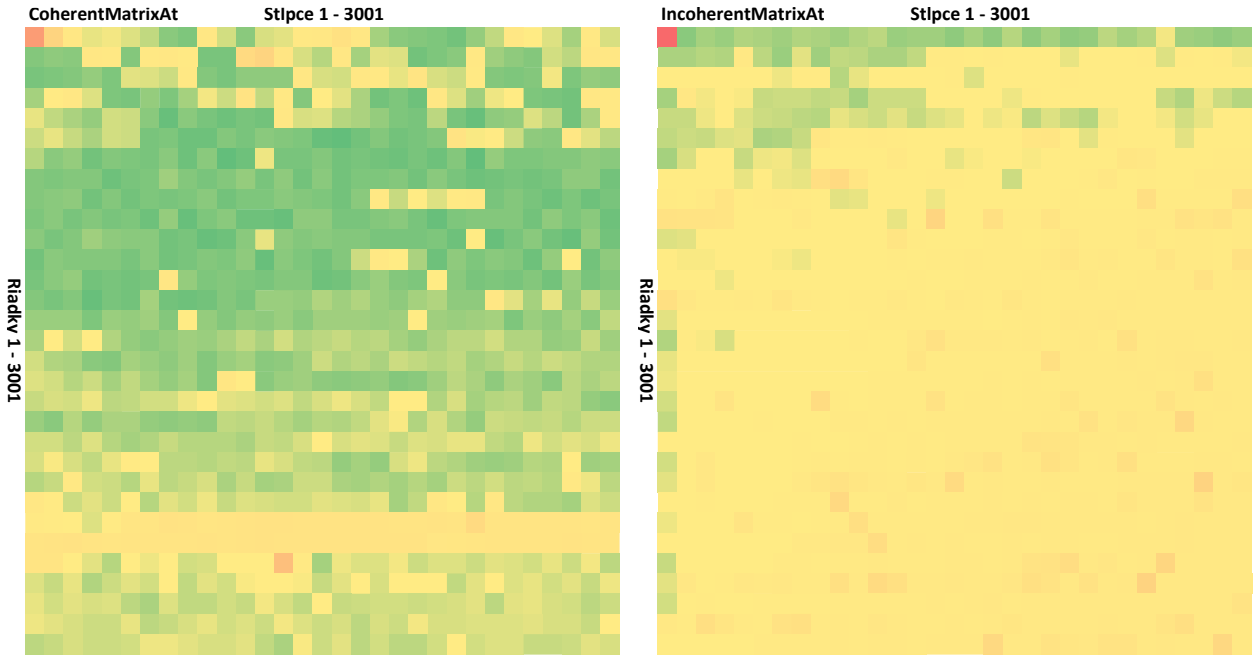
Porovnanie Assign()

Heat mapa je použitá na obidve matice globálne, ale len na metódu assign().
Tak ako sme už v predošlých scenároch spozorovali, aj tu má vplyv aj počet riadkov aj počet stĺpcov, avšak pri nesúvislej pamäti má väčší vplyv počet riadkov.

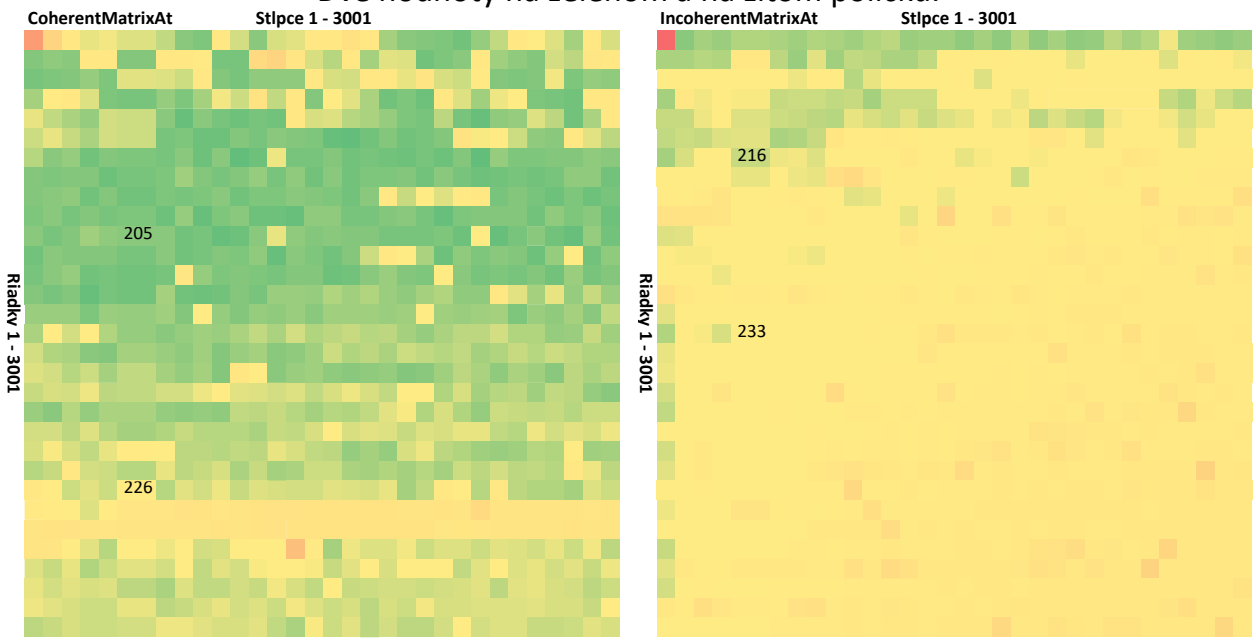


Porovnanie At()

Heat mapa je použitá na obidve matice globálne, ale len na metódu at().
Je vidno, že pri súvislej pamäti je heat mapa na riadkoch veľmi málo farebne rozložená, to znamená, že pri stĺpce nie sú významne a pri riadkoch tam je zmena ktorá skáče z horšej rýchlosti na lepšiu a znova na horšiu. Keď si však pridáme hodnotu na žltom a na zelenšom políčku, zistíme, že rozdiel je minimálny a čas mohol byť ovplyvnený aplikáciami na pozadí. Pri nesúvislej pamäti je to rozloženie takmer rovnaké po celý čas.



Dve hodnoty na zelenom a na žltom políčku.



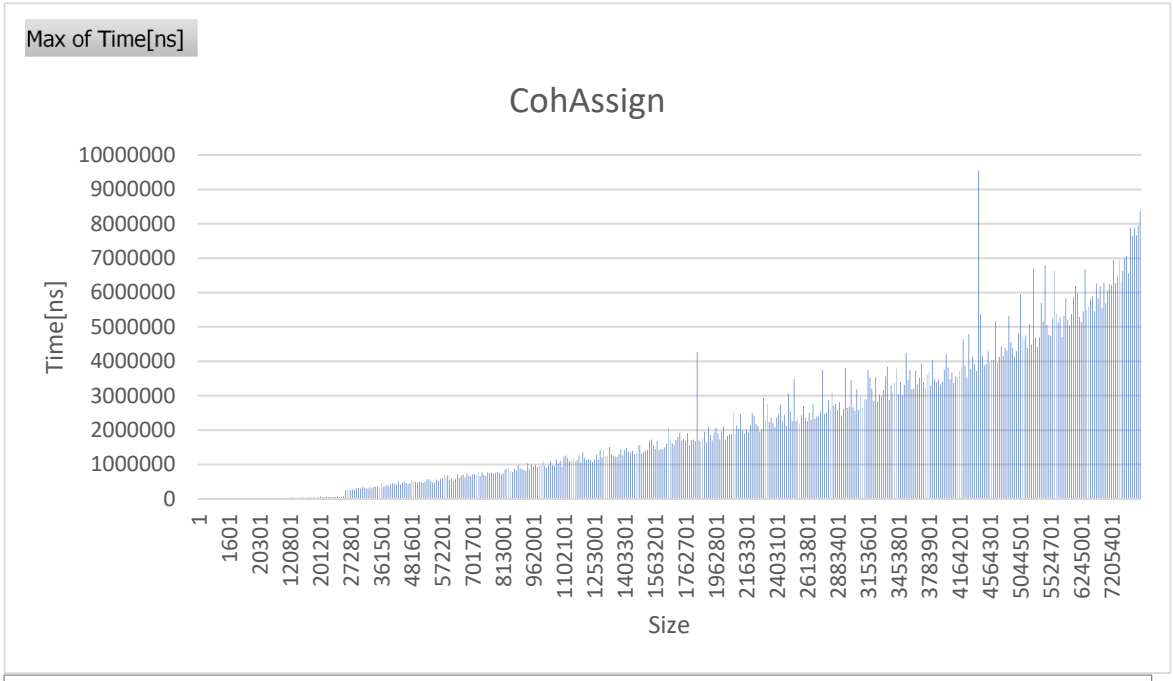
Záver

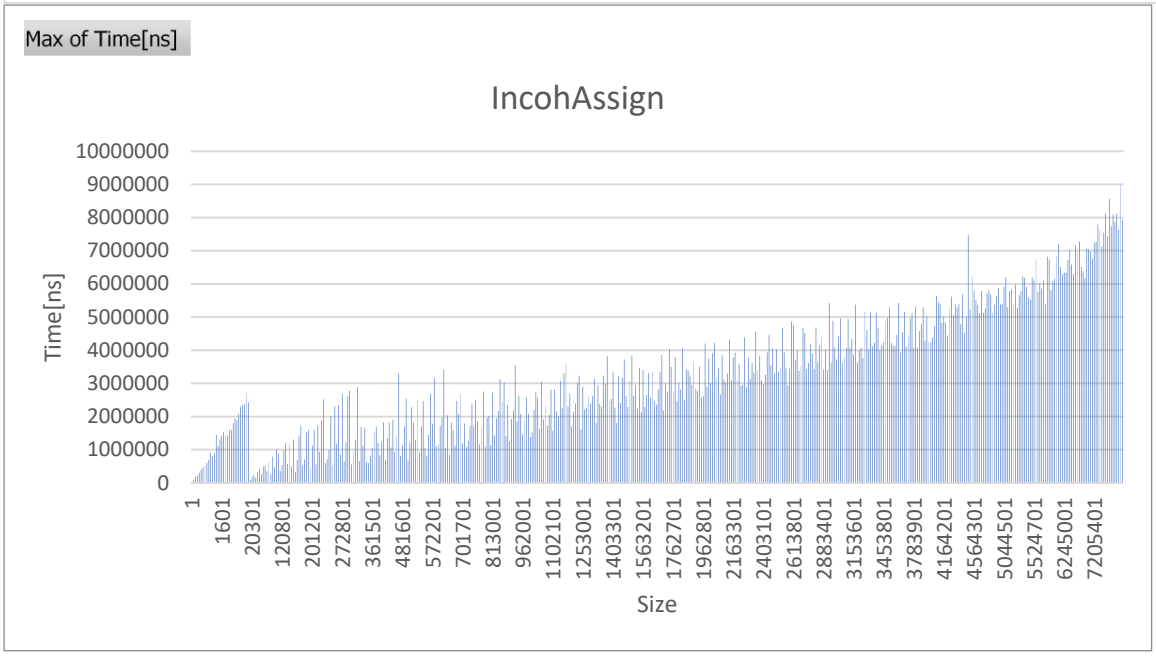
Z heat máp môžeme vyčítať, že

- Assign v matici so súvislou pamäťou je rozložený rovnomerne bez ohľadu na veľkosť riadkov a stĺpcov.
- Assign v matici s nesúvislou pamäťou je rozloženie tiež rovnomerné, avšak väčší vplyv na rýchlosť majú riadky.
- At v matici so súvislou pamäťou vyzerá ako keby mali vplyv na rýchlosť riadky, avšak rozdiel vo farbách je len pár nanosekúnd a cyklus sa robil najprv cez riadky a až potom cez stĺpce, tým pádom aplikácie na pozadí mohli ovplyvniť riadky a nie stĺpce danej heat mapy.
- At v matici s nesúvislou pamäťou nám ukazuje, že pokiaľ mám jeden riadok alebo stĺpec, rýchlosť je lepšia ako keď tam riadky alebo stĺpce pridávam. Avšak to platí len pre prvý riadok a prvý stĺpec. Potom vidíme mierne sfarbenie do pravého dolného rohu, čo značí, že počet riadkov a počet stĺpcov majú taký istý ale veľmi mierny vplyv na rýchlosť.

Odhad hornej asymptotickej zložitosti

Na odhad som si roztiahol maticu s časom kde riadky boli počet riadkov v matici a stĺpce počet stĺpcov v matici do vektora podľa riadkov a spravil dva grafy na metódu Assign() a metódu At().





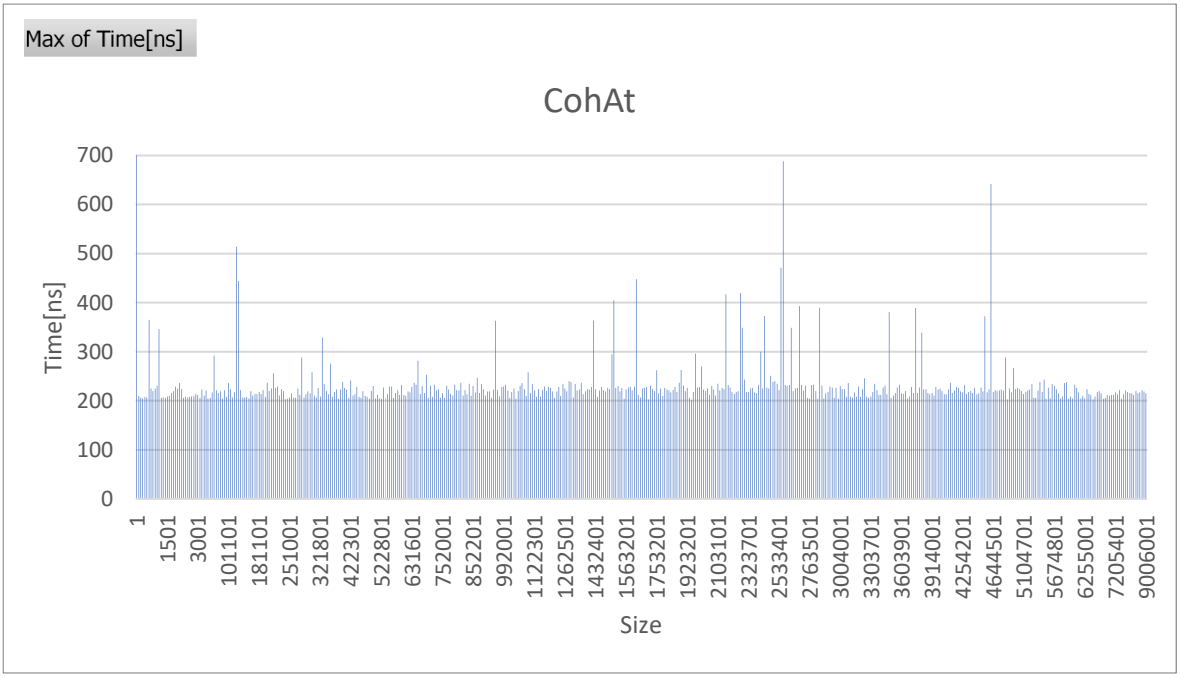
Podľa grafov vidím, že zložitosť metódy assign je kvadratická a matica so súvislou pamäťou je o trošičku rýchlejšia ako matica s nesúvislou pamäťou. Assign metóda má teda zložitosť $O(n^2)$.

Podľa excelu som si určil trend a pripočítal ku nemu konštantu aby som ho posunul vyššie a vyšli mi takéto výsledky.

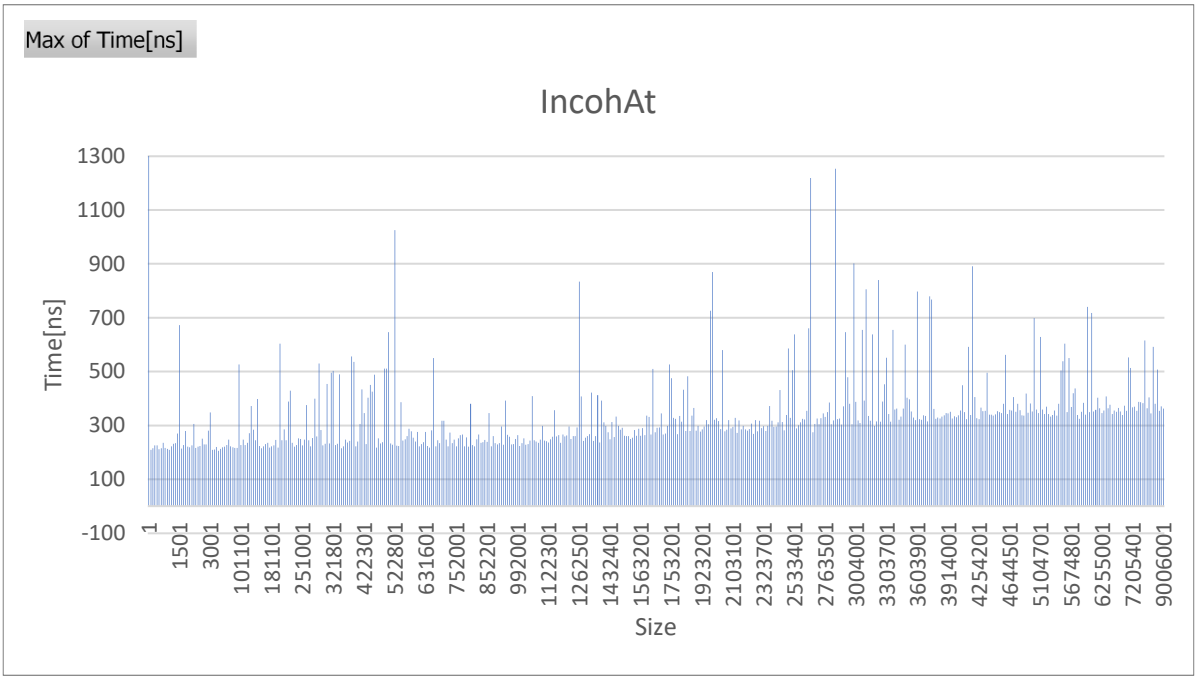
Horný odhad pre súvislú pamäť: $y = 26{,}771x^2 - 213{,}84x + 1\ 000\ 000$

Horný odhad pre nesúvislú pamäť: $y = 20{,}355x^2 + 2465{,}9x + 2\,000\,000$

Tu som však musel odseknúť začiatok až po veľkosť 20301.



Podľa grafu vidím, že zložitosť metódy at v súvislej pamäti je konštantná, teda zložitosť je O(1). Mohol by som odhadnúť horný asymptoticky odhad na 700 nanosekúnd, pretože najvyššia nameraná hodnota tesne pod 700 a nepredpokladám, že túto hodnotu presiahnem.



Podľa grafu vidím, že rýchlosť pomaly klesá, tým pádom je zložitosť lineárna O(n).
A podľa excel trendu som určil horný odhad na y = 0,2108x + 900, pretože chcem začať na 900, keďže sú tam nejaké výkyvy.