Matriisilaskimen laskuoperaatioiden toimivuutta on testattu 73 yksikkötestillä, ja suorituskykytestausta on tehty manuaalisilla syötteillä. Käyttöliittymää on testattu manuaalisesti eri syötteillä ja jokaisen gui-matriisin kohdalla toiminnallisuus pelaa niin kuin pitääkin. Virhe-popupit käyttöliittymälle kehitin tämän manuaalisen testauksen seurauksena. Myös Matriisilaskin-luokan toiminnallisuus tuli testattua käyttöliittymän testauksen yhteydessä.

Suorituskykytestaus toteutettiin ottamalla mitattavasta suorituksesta kymmenen aikaa, joista kaksi pienintä ja suurinta hylättiin ja jäljelle jääneistä otettiin keskiarvo. Manuaaliset syötteet erosivat laskuoperaatiokohtaisesti. Kaikki suorituskykytestit voidaan toistaa manuaalisesti MatriisienGeneroija-luokan metodien ja laskuoperaatioluokkien avulla.

Testaustulokset: Tavallinen matriisin kertolasku

Manuaaliset syötteet kaikilla kertolaskutesteillä olivat ykkösillä täytettyjä neliömatriiseja, koska tulosten validisuus on tällöin helppo tarkastaa (jokainen alkio sama kuin tulomatriisin koko, esim. 100x100 alkiot ovat 100). Koot valittiin kahden potensseiksi, jotta vertailu Strassenin algoritmin kanssa olisi mahdollisimman yhdenvertainen (Strassen täyttää matriisit aina seuraavaan kahden potenssiin, esim. 100->128). Suoritusajat:

128x128 matriisien tulo: 10ms

256x256 matriisien tulo: 55ms

512x512 matriisien tulo: 606.67ms

1024x1024 matriisien tulo: 11600ms

2048x2048 matriisien tulo: 125570.5ms

4096x4096 matriisien tulo: 1257608ms (suorituksen keston takia otin vapauden mitata vain yhden hyvin suuntaa antavan ajan)

Testaustulokset: Matriisin kertolasku täysin rekursiivisella Strassenilla

Lähdin aluksi testaamaan täysin rekursiivisen Strassenin suorituskykyä, joka pilkkoo lohkomatriisit rekursiivisesti aina 1x1 matriiseihin asti. Suoritusajat:

128x128 matriisien tulos Strassenilla: 137.33ms

256x256 matriisien tulo Strassenilla: 837.5ms

512x512 matriisien tulo Strassenilla: 5525ms

1024x1024 matriisien tulo Strassenilla: 38400ms

2048x2048 matriisien tulo Strassenilla: 268612ms

4096x4096 matriisien tulo Strassenilla: 1859641ms (suorituksen keston takia otin vapauden mitata vain yhden hyvin suuntaa antavan ajan)

Tuloksiin pettyneenä päätin hieman optimoida Strassenin algoritmia ja muutaman testauksen jälkeen huomasin 32x32 kokoisen matriisin olevan optimaalisin lohkomatriisikoko siirtyä tavalliseen matriisin kertolaskuun.

Testaustulokset: Matriisin kertolasku optimoidulla Strassenilla

Optimoitu Strassen ei ole täysin rekursiivinen 1x1 matriiseihin asti, vaan antaa tavallisen matriisin kertolaskun suorittaa 32x32 tai sitä pienempien matriisien kertolaskun, koska se on pienillä matriisikooilla paljon Strassenia nopeampi. Suoritusajat:

128x128 matriisien tulo optimoidulla Strassenilla: 20ms

256x256 matriisien tulo optimoidulla Strassenilla: 58.33ms

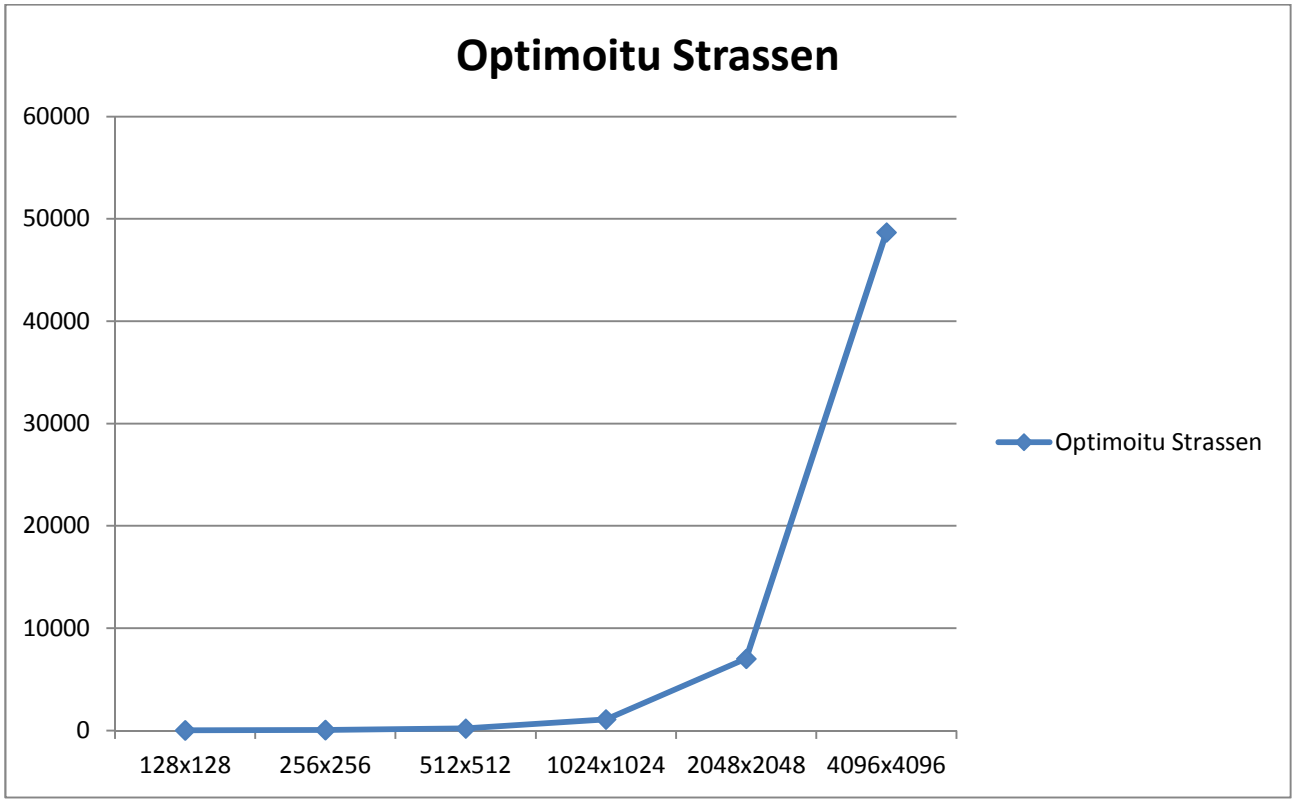
512x512 matriisien tulo optimoidulla Strassenilla: 200ms

1024x1024 matriisien tulo optimoidulla Strassenilla: 1098.83ms

2048x2048 matriisien tulo optimoidulla Strassenilla: 7017ms

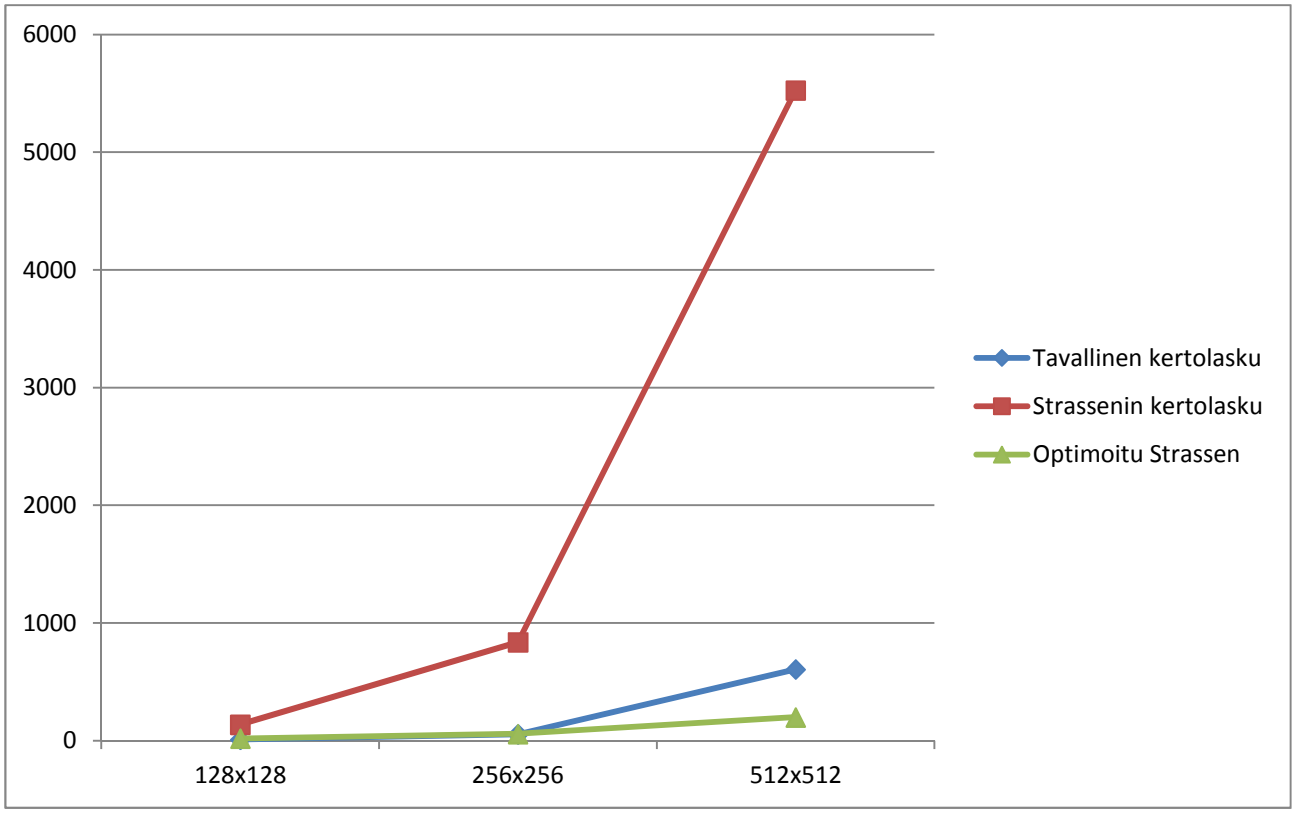
4096x4096 matriisien tulo optimoidulla Strassenilla: 48684.5ms

Graafinen esitys matriisin kertolaskun suorituskykytuloksille



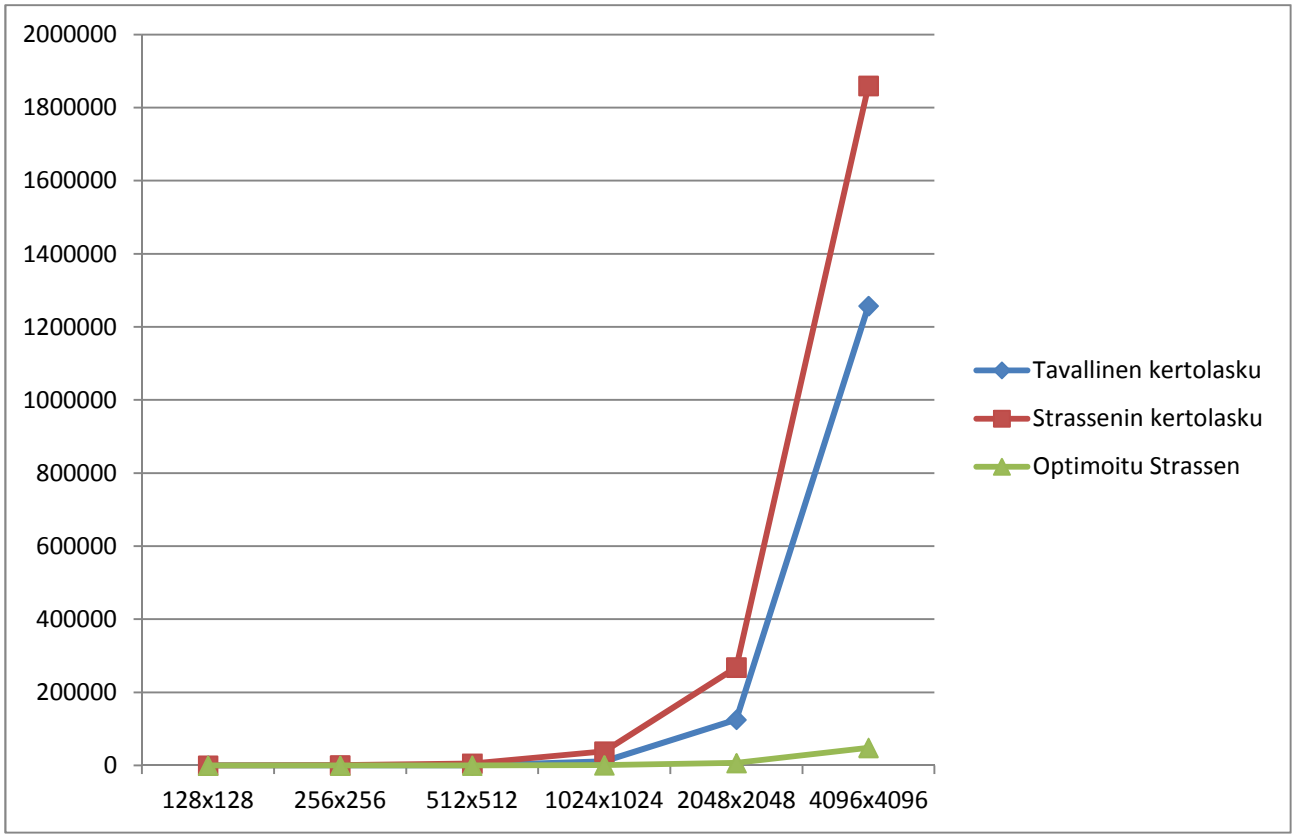
Kuva 1. Optimoidun Strassenin tulokset. Pystyakseli millisekunteina.

Kuvasta 1. huomataan, että optimoitu Strassen näyttää toteuttavan hyvin keskimäärin $O(n^{2.8074})$ aikavaativuutta.



Kuva 2. Pienempien matriisikokojen erot. Pystyakseli millisekunteina.

Kuten kuvasta 2. huomaa, täysin rekursiivinen Strassen on pienillä matriisikooilla paljon tavallista kertolaskua hitaampi. Toisinkuin täysin rekursiivinen Strassen, optimoitu pysyy pienemmillä matriisikooilla hyvin tavallisen kertolaskun perässä ja alkaa ottaa isompaa pesäeroa, kun päästään yli 256x256 matriiseihin.



Kuva 3. Suorituskykytestien tulokset kokonaisuutena. Pystyakseli millisekunteina.

Kuvasta 3. huomataan, että tavallinen kertolasku on täysin rekursiivista Strassenia nopeampi vielä 4096x4096 matriisikooilla, mutta vaikuttaisi siltä, että tilanne on tasoittumaan päin mitä suurempiin matriisikokoihin päästään. Optimoitu Strassen on taas omaa luokkaansa ja 4096x4096 matriisien kertolaskut lasketaan 49 sekuntiin, kun muilla kestää 20 minuutista 30 minuuttiin. Näin valtavia suorituskykyeroja osaltaan selittää varmasti Javan omat sisäiset rajoitukset, esimerkiksi 8192x8192 matriisikooilla Javan heap space otti jo vastaan ja antoi out of memory erroria.

Testaustulokset: Matriisin determinantti

Manuaaliset syötteet kaikilla determinanttitesteillä olivat yksikkömatriiseja, koska tulosten validisuus on tällöin helppo tarkastaa (determinantti on yksi).

200x200 determinantti: 15ms

400x400 determinantti: 30ms

600x600 determinantti: 70ms

800x800 determinantti: 150ms

1000x1000 determinantti: 311.67ms

1200x1200 determinantti: 573.33ms

1400x1400 determinantti: 930ms

1600x1600 determinantti: 1411.67ms

1800x1800 determinantti: 1986.67ms

2000x2000 determinantti: 2701.67ms

3000x3000 determinantti: 9221.67ms

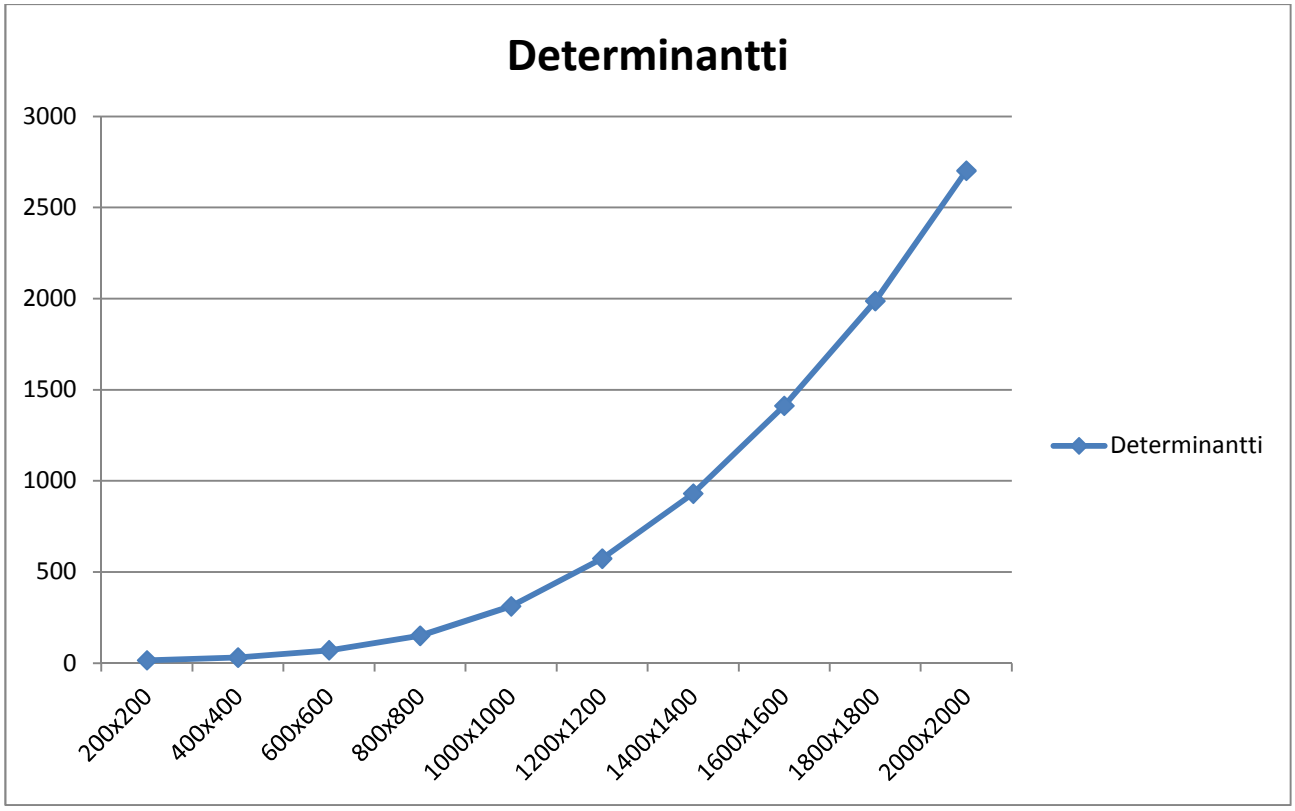
4000x4000 determinantti: 21440.67ms

5000x5000 determinantti: 42200.33ms

6000x6000 determinantti: 73215.67ms

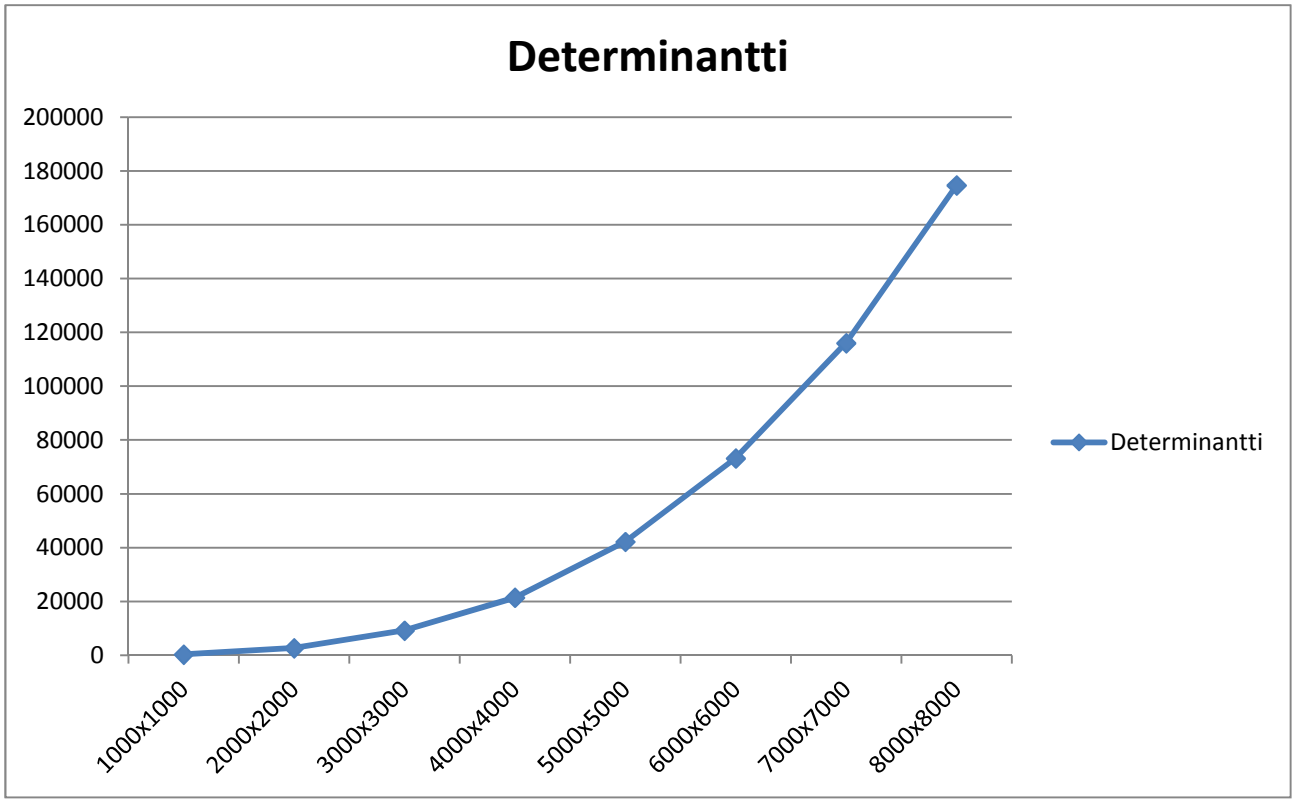
7000x7000 determinantti: 116036.67ms

8000x8000 determinantti: 174647.67ms



Kuva 4. Pienempien matriisikokojen erot. Pystyakseli millisekunteina.

Kuvasta 4. huomataan, että pienempien matriisikokojen perusteella determinantti-algoritmi näyttää toteuttavan hyvin kuutiollista aikavaativuutta.



Kuva 5. Isompien matriisikokojen erot. Pystyakseli millisekunteina.

Kuvasta 5. huomataan, että isompien matriisikokojen perusteella Javan sisäinen rakenne ei näytä hidastavan determinantin laskemista oleellisesti. 10000x10000 determinantti meni vielä alle kuuteen minuuttiin, mutta 11000x11000 matriisilla tuli jälleen Javan heap space vastaan.

Testaustulokset: Käänteismatriisin approksimointi

Manuaaliset syötteet kaikilla käänteismatriisitesteillä olivat yksikkömatriiseja, koska tulosten validisuus on tällöin helppo tarkastaa (käänteismatriisi on myös yksikkömatriisi). Ajanottamisessa ei otettu huomioon kääntyvyyteen liittyvää tarkastusta, joka olisi johtanut determinantin laskemiseen, vaan aika otettiin Kaanteismatriisi-luokan invertoi-metodin alusta loppuun.

200x200 käänteismatriisi: 40ms

400x400 käänteismatriisi: 131.67ms

600x600 käänteismatriisi: 600ms

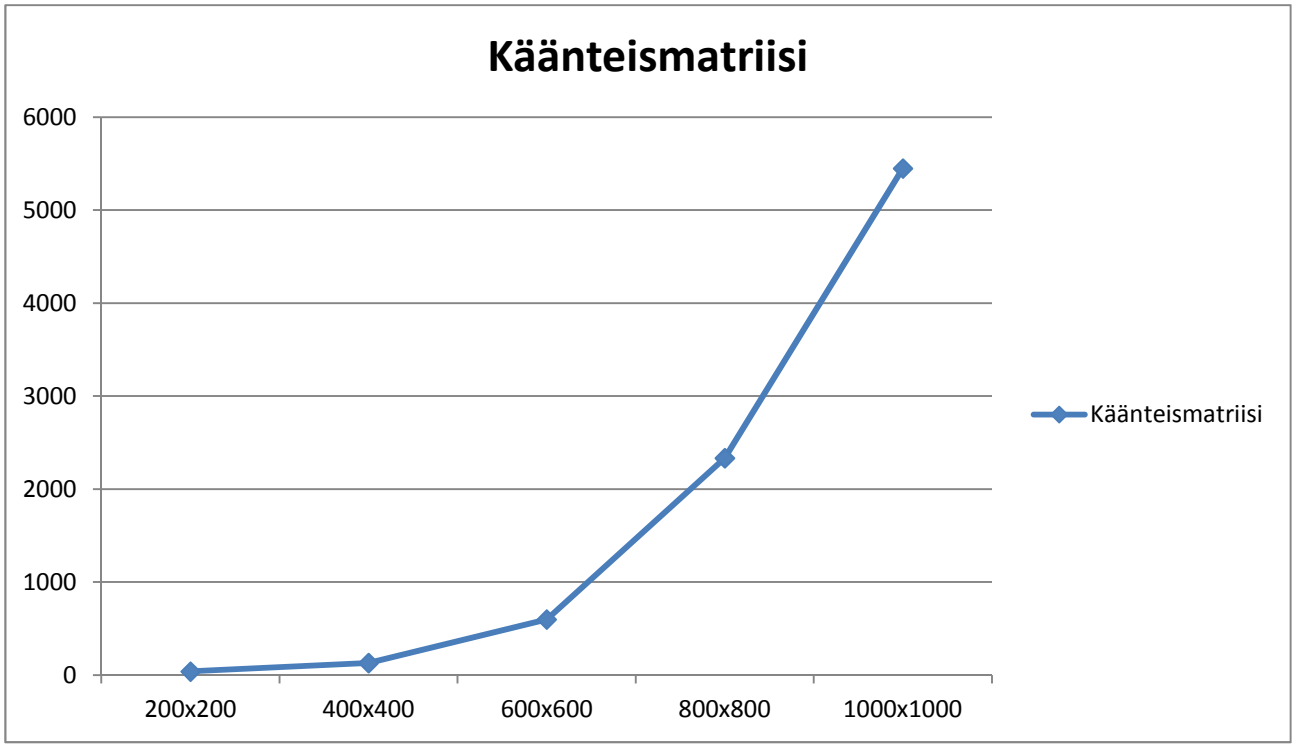
800x800 käänteismatriisi: 2335ms

1000x1000 käänteismatriisi: 5450ms

2000x2000 käänteismatriisi: 61683.33ms

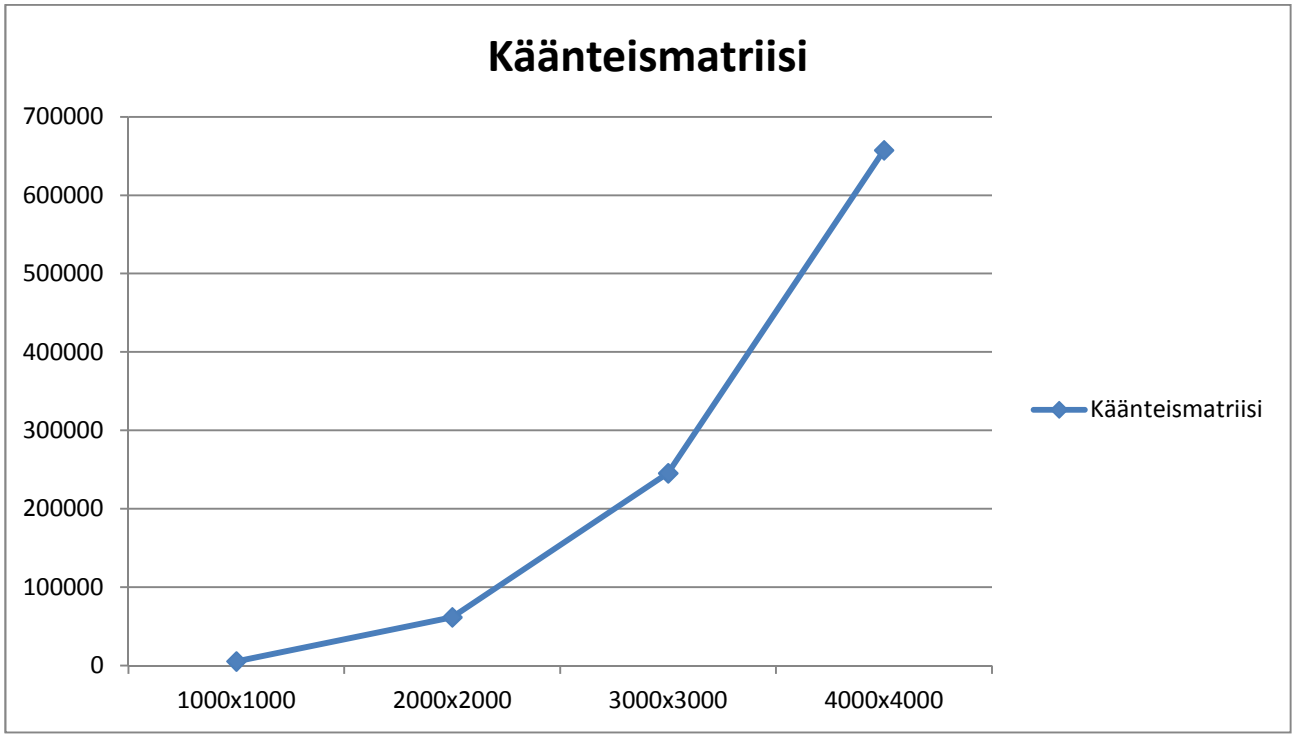
3000x3000 käänteismatriisi: 245493ms

4000x4000 käänteismatriisi: 657431ms



Kuva 6. Pienempien matriisikokojen erot. Pystyakseli millisekunteina.

Kuvasta 6. huomataan, että pienempien matriisikokojen perusteella käänteismatriisin approksimointi näyttää toteuttavan hyvin kuutiollista aikavaativuutta.



Kuva 7. Isompien matriisikokojen erot. Pystyakseli millisekunteina.

Kuvasta 7. huomataan, että isompien matriisikokojen perusteella Javan sisäinen rakenne ei näytä hidastavan käänteismatriisin approksimointia oleellisesti. 11000x11000 matriisille tuli jälleen Javan

heap space vastaan, mutta 10000x10000 matriisia antoi ajaa päälle minuutin ennen kuin lopetin sen itse ennenaikaisesti sen odotetun keston takia.

Testaustulokset: Potenssiin korottaminen neliöimällä

Manuaaliset syötteet kaikilla potenssiin korottamistesteillä olivat 1024x1024 kokoisia yksikkömatriiseja. Ykkösillä täytettyjen matriisien käyttö olisi muuten suotavampaa, mutta alkioiden suuruus kasvaa räjähdysmäisesti potenssin kasvaessa tämän kokoisilla matriiseilla, joten tulosten validisuutta on vaikeampi tarkastaa.

$[1024x1024]^2$ potenssiin korottaminen neliöimällä: 1120ms

$[1024x1024]^3$ potenssiin korottaminen neliöimällä: 2170ms

$[1024x1024]^4$ potenssiin korottaminen neliöimällä: 2160ms

$[1024x1024]^5$ potenssiin korottaminen neliöimällä: 3130ms

$[1024x1024]^6$ potenssiin korottaminen neliöimällä: 3130ms

$[1024x1024]^7$ potenssiin korottaminen neliöimällä: 4150ms

$[1024x1024]^8$ potenssiin korottaminen neliöimällä: 3180ms

$[1024x1024]^9$ potenssiin korottaminen neliöimällä: 4230ms

$[1024x1024]^{10}$ potenssiin korottaminen neliöimällä: 4210ms

Vertailun vuoksi sama testi toistettiin käyttäen pelkästään Strassenin kertolaskuja.

$[1024x1024]^2$ potenssiin korottaminen Strassenilla: 1090ms

$[1024x1024]^3$ potenssiin korottaminen Strassenilla: 2150ms

$[1024x1024]^4$ potenssiin korottaminen Strassenilla: 3180ms

$[1024x1024]^5$ potenssiin korottaminen Strassenilla: 4120ms

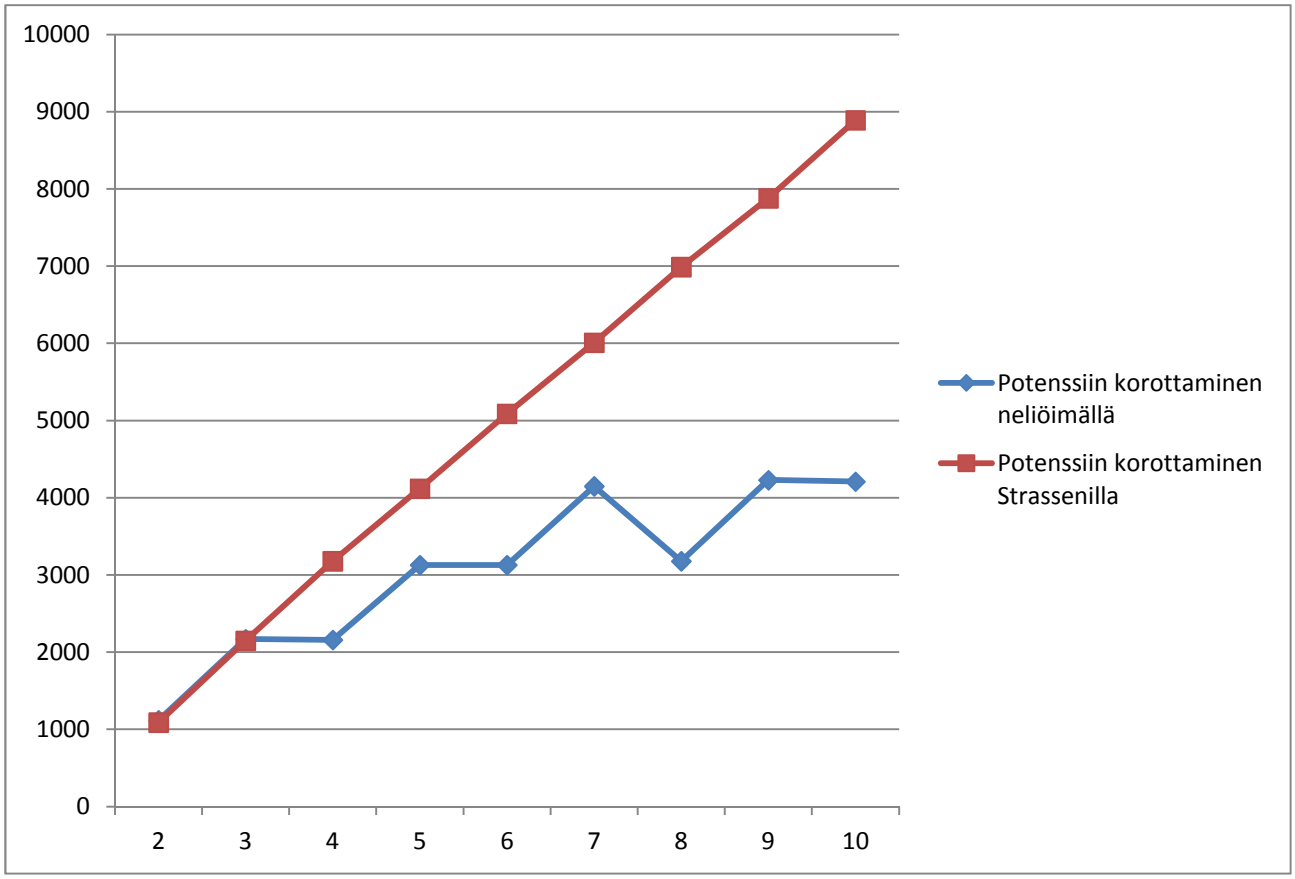
$[1024x1024]^6$ potenssiin korottaminen Strassenilla: 5090ms

$[1024x1024]^7$ potenssiin korottaminen Strassenilla: 6010ms

$[1024x1024]^8$ potenssiin korottaminen Strassenilla: 6990ms

$[1024x1024]^9$ potenssiin korottaminen Strassenilla: 7880ms

$[1024x1024]^{10}$ potenssiin korottaminen Strassenilla: 8890ms



Kuva 8. Potenssiin korottamisen vertailu. Vaaka-akselilla potenssit ja pystyakselilla millisekuntit.

Kuvasta 8. huomataan, että potenssiin korottaminen Strassenilla toteutuu potenssiin nähden lineaarisesti ja potenssiin korottaminen neliöimällä toteutuu potenssiin nähden logaritmisesti juuri niin kuin pitääkin. Ero näkyy selvimmin suuremmilla potensseilla, esim. $[1024x1024]^{1000}$:ssa kestää noin 13 sekuntia neliöimällä, kun taas Strassenilla siinä kuluu päälle 15 minuuttia.