

Algorytmy macierzowe

Laboratorium 2

Marcel Duda, Jan Gawroński

17.11.2025

1 Rekurencyjne odwracanie macierzy

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad (1)$$

$$A^{-1} = \begin{bmatrix} A_{11}^{-1} + A_{11}^{-1} A_{12} S_{22}^{-1} A_{21} A_{11}^{-1} & -A_{11}^{-1} A_{12} S_{22}^{-1} \\ -S_{22}^{-1} A_{21} A_{11}^{-1} & S_{22}^{-1} \end{bmatrix} \quad (2)$$

gdzie $S_{22} = A_{22} - A_{21} A_{11}^{-1} A_{12}$, jeśli A jest macierzą 1×1 , to $A^{-1} = \left[\frac{1}{a_{11}} \right]$.

1.1 Implementacja

```
1 Matrix inverse(const Matrix &A, std::unique_ptr<IMnozenie> &
2     multImpl) {
3     if (rows(A) == 1) {
4         Matrix invA = zeroMatrix(1, 1);
5         invA[0][0] = 1.0 / A[0][0];
6         opCounterAdd({0, 0, 0, 1});
7         return invA;
8     }
9     if (rows(A) % 2 == 0) {
10        memCounterEnterCall(rows(A), cols(A), 3);
11        int halfSize = rows(A) / 2;
12        Matrix invA11 = inverse(subMatrix(A, 0, 0, halfSize,
13            halfSize), multImpl);
14        Matrix A12 = subMatrix(A, 0, halfSize, halfSize, halfSize)
15            ;
16        Matrix A21 = subMatrix(A, halfSize, 0, halfSize, halfSize)
17            ;
18        Matrix A22 = subMatrix(A, halfSize, halfSize, halfSize,
19            halfSize);
```

```

19     Matrix invS22 = inverse(A22 - multImpl->multiply(A21, T1),
20                               multImpl);
21
22     Matrix T3 = multImpl->multiply(T1, invS22);
23
24     Matrix B11 = invA11 + multImpl->multiply(T3, T2);
25     Matrix B12 = negate(T3);
26     Matrix B21 = negate(multImpl->multiply(invS22, T2));
27     Matrix B22 = invS22;
28
29     memCounterExitCall(rows(A), cols(A), 3);
30     return combine(B11, B12, B21, B22);
31 } else {
32     Matrix A_padded = pad(A, rows(A) + 1, cols(A) + 1);
33     A_padded[rows(A)][cols(A)] = 1.0;
34     Matrix inv_padded = inverse(A_padded, multImpl);
35 }
36 }
```

1.2 Wykresy

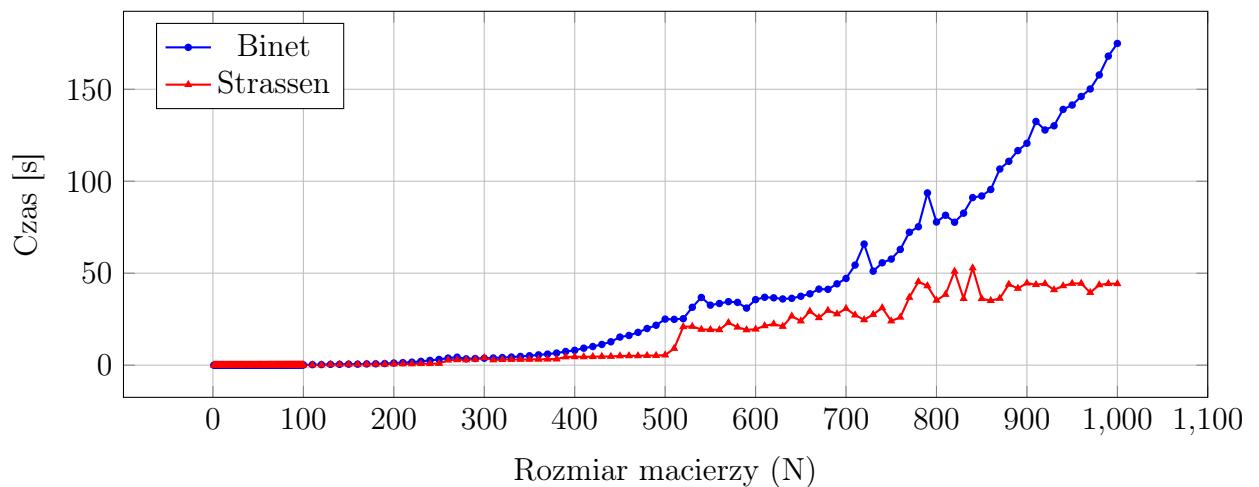


Figure 1.2.1: Czas działania (Binet vs Strassen)

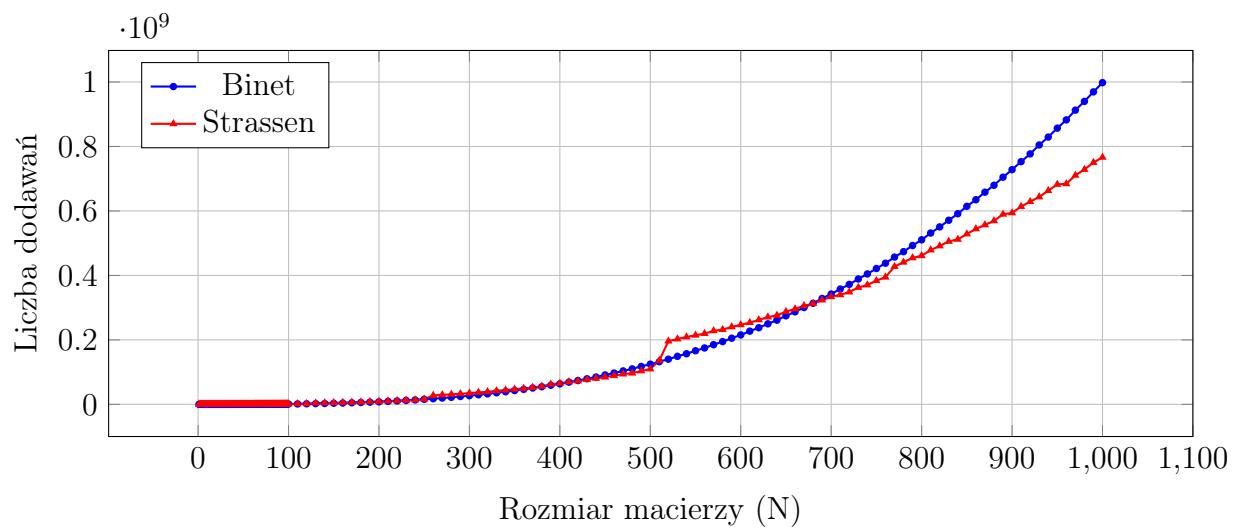


Figure 1.2.2: Porównanie liczby operacji dodawania

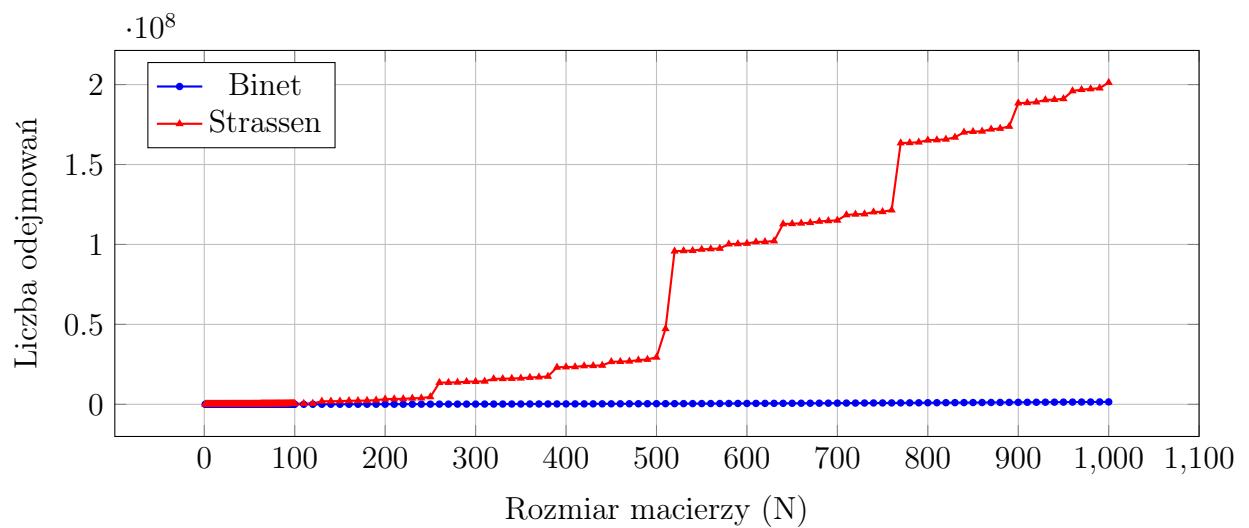


Figure 1.2.3: Porównanie liczby operacji odejmowania

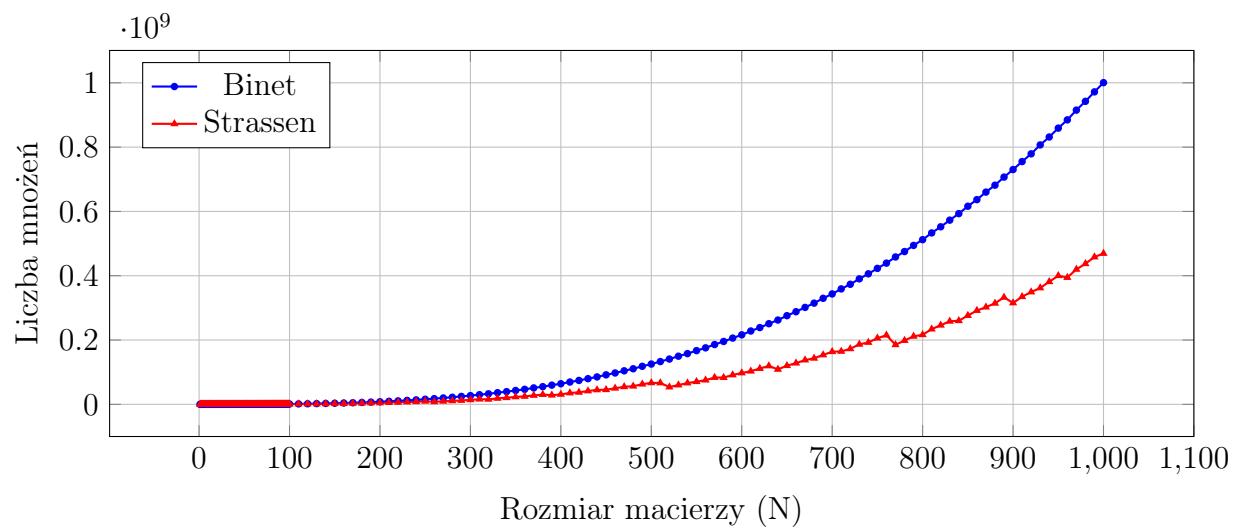


Figure 1.2.4: Porównanie liczby operacji mnożenia

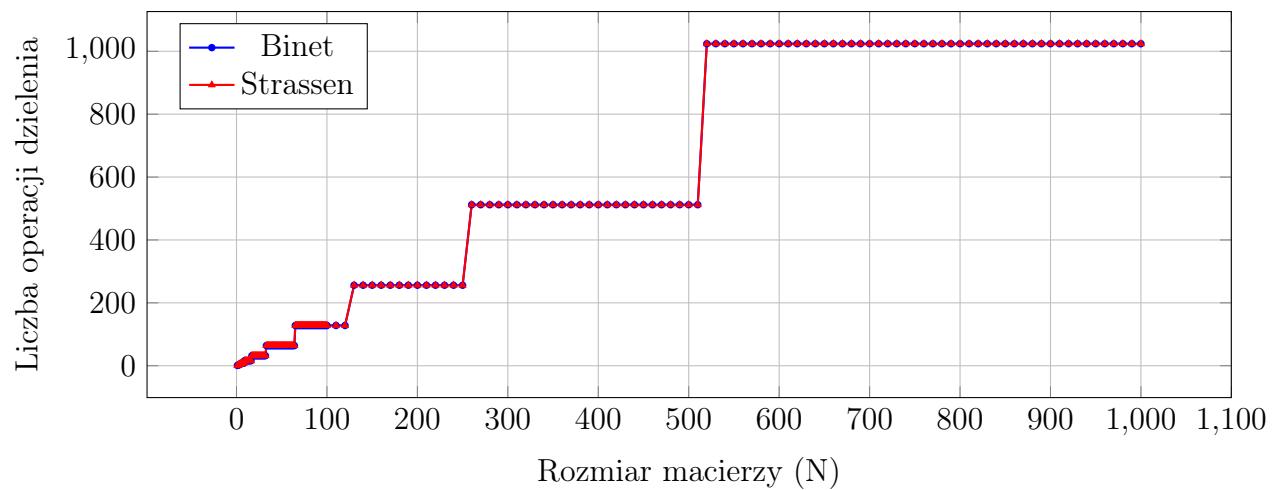


Figure 1.2.5: Porównanie liczby operacji dzielenia

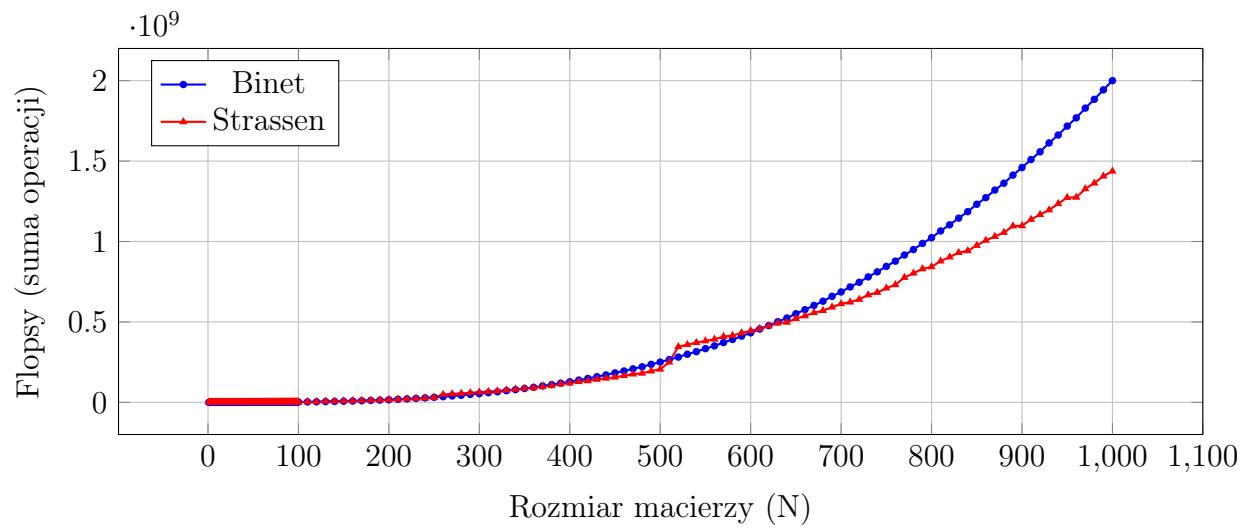


Figure 1.2.6: Porównanie liczby operacji zmiennoprzecinkowych (flops)

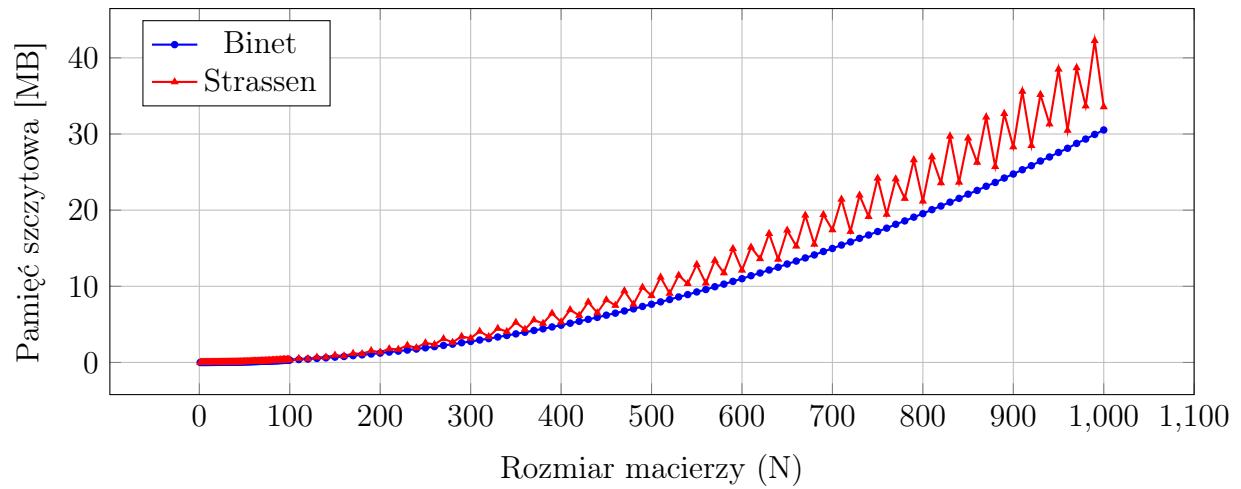


Figure 1.2.7: Porównanie zużycia pamięci

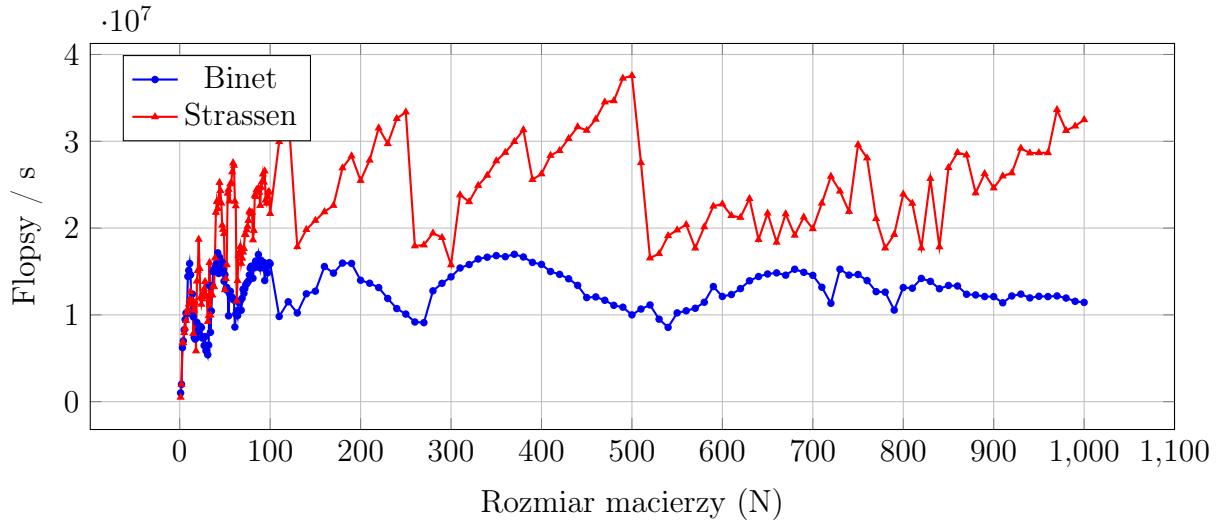


Figure 1.2.8: Przepustowość (flops / s) porównanie

2 Eliminacja Gaussa

Dane A, b:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad (3)$$

$$b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad (4)$$

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad (5)$$

$$A_{11}x_1 + A_{12}x_2 = b_1 \quad (6)$$

$$A_{21}x_1 + A_{22}x_2 = b_2 \quad (7)$$

$$A_{11} = L_{11}U_{11} \quad (8)$$

$$S = A_{22} - A_{21}U_{11}^{-1}L_{11}^{-1}A_{12} = L_S U_S \quad (9)$$

$$C = \begin{bmatrix} U_{11} & L_{11}A_{12} \\ 0 & U_S \end{bmatrix} \quad (10)$$

$$c = \begin{bmatrix} L_{11}^{-1}b_1 \\ L_S^{-1}(b_2 - A_{21}U_{11}^{-1}L_{11}^{-1}b_1) \end{bmatrix} \quad (11)$$

Zwróć C, c.

2.1 Implementacja

```

1 std::pair<Matrix, Matrix> GaussElimination(const Matrix &A, const
2     Matrix &b, std::unique_ptr<IMnozenie> &multImpl) {
3     if (rows(A) == 1) {
4         return {A, b};
5     }
6     Matrix L(rows(A), rows(A));
7     Matrix U(rows(A), rows(A));
8     Vector b1(rows(A));
9     Vector b2(rows(A));
10    Vector c1(rows(A));
11    Vector c2(rows(A));
12
13    for (int i = 0; i < rows(A); ++i) {
14        for (int j = 0; j < i; ++j) {
15            L(i, j) = A(i, j);
16        }
17        for (int j = i + 1; j < rows(A); ++j) {
18            U(i, j) = A(i, j);
19        }
20        b1(i) = b(i);
21    }
22
23    for (int i = 0; i < rows(A); ++i) {
24        for (int j = 0; j < i; ++j) {
25            U(j, i) = 0;
26        }
27    }
28
29    for (int i = 0; i < rows(A); ++i) {
30        for (int j = i + 1; j < rows(A); ++j) {
31            L(i, j) = U(i, j) / U(i, i);
32            U(j, i) = 0;
33        }
34    }
35
36    for (int i = 0; i < rows(A); ++i) {
37        for (int j = 0; j < i; ++j) {
38            U(j, i) = 0;
39        }
40    }
41
42    for (int i = 0; i < rows(A); ++i) {
43        for (int j = 0; j < i; ++j) {
44            L(i, j) = 0;
45        }
46    }
47
48    for (int i = 0; i < rows(A); ++i) {
49        for (int j = 0; j < i; ++j) {
50            L(i, j) = U(i, j);
51        }
52    }
53
54    for (int i = 0; i < rows(A); ++i) {
55        for (int j = 0; j < i; ++j) {
56            U(j, i) = 0;
57        }
58    }
59
60    for (int i = 0; i < rows(A); ++i) {
61        for (int j = 0; j < i; ++j) {
62            L(i, j) = 0;
63        }
64    }
65
66    for (int i = 0; i < rows(A); ++i) {
67        for (int j = 0; j < i; ++j) {
68            L(i, j) = U(i, j);
69        }
70    }
71
72    for (int i = 0; i < rows(A); ++i) {
73        for (int j = 0; j < i; ++j) {
74            U(j, i) = 0;
75        }
76    }
77
78    for (int i = 0; i < rows(A); ++i) {
79        for (int j = 0; j < i; ++j) {
80            L(i, j) = 0;
81        }
82    }
83
84    for (int i = 0; i < rows(A); ++i) {
85        for (int j = 0; j < i; ++j) {
86            L(i, j) = U(i, j);
87        }
88    }
89
90    for (int i = 0; i < rows(A); ++i) {
91        for (int j = 0; j < i; ++j) {
92            U(j, i) = 0;
93        }
94    }
95
96    for (int i = 0; i < rows(A); ++i) {
97        for (int j = 0; j < i; ++j) {
98            L(i, j) = 0;
99        }
100   }
101
102   for (int i = 0; i < rows(A); ++i) {
103       for (int j = 0; j < i; ++j) {
104           L(i, j) = U(i, j);
105       }
106   }
107
108   for (int i = 0; i < rows(A); ++i) {
109       for (int j = 0; j < i; ++j) {
110           U(j, i) = 0;
111       }
112   }
113
114   for (int i = 0; i < rows(A); ++i) {
115       for (int j = 0; j < i; ++j) {
116           L(i, j) = 0;
117       }
118   }
119
120   for (int i = 0; i < rows(A); ++i) {
121       for (int j = 0; j < i; ++j) {
122           L(i, j) = U(i, j);
123       }
124   }
125
126   for (int i = 0; i < rows(A); ++i) {
127       for (int j = 0; j < i; ++j) {
128           U(j, i) = 0;
129       }
130   }
131
132   for (int i = 0; i < rows(A); ++i) {
133       for (int j = 0; j < i; ++j) {
134           L(i, j) = 0;
135       }
136   }
137
138   for (int i = 0; i < rows(A); ++i) {
139       for (int j = 0; j < i; ++j) {
140           L(i, j) = U(i, j);
141       }
142   }
143
144   for (int i = 0; i < rows(A); ++i) {
145       for (int j = 0; j < i; ++j) {
146           U(j, i) = 0;
147       }
148   }
149
150   for (int i = 0; i < rows(A); ++i) {
151       for (int j = 0; j < i; ++j) {
152           L(i, j) = 0;
153       }
154   }
155
156   for (int i = 0; i < rows(A); ++i) {
157       for (int j = 0; j < i; ++j) {
158           L(i, j) = U(i, j);
159       }
160   }
161
162   for (int i = 0; i < rows(A); ++i) {
163       for (int j = 0; j < i; ++j) {
164           U(j, i) = 0;
165       }
166   }
167
168   for (int i = 0; i < rows(A); ++i) {
169       for (int j = 0; j < i; ++j) {
170           L(i, j) = 0;
171       }
172   }
173
174   for (int i = 0; i < rows(A); ++i) {
175       for (int j = 0; j < i; ++j) {
176           L(i, j) = U(i, j);
177       }
178   }
179
180   for (int i = 0; i < rows(A); ++i) {
181       for (int j = 0; j < i; ++j) {
182           U(j, i) = 0;
183       }
184   }
185
186   for (int i = 0; i < rows(A); ++i) {
187       for (int j = 0; j < i; ++j) {
188           L(i, j) = 0;
189       }
190   }
191
192   for (int i = 0; i < rows(A); ++i) {
193       for (int j = 0; j < i; ++j) {
194           L(i, j) = U(i, j);
195       }
196   }
197
198   for (int i = 0; i < rows(A); ++i) {
199       for (int j = 0; j < i; ++j) {
200           U(j, i) = 0;
201       }
202   }
203
204   for (int i = 0; i < rows(A); ++i) {
205       for (int j = 0; j < i; ++j) {
206           L(i, j) = 0;
207       }
208   }
209
210   for (int i = 0; i < rows(A); ++i) {
211       for (int j = 0; j < i; ++j) {
212           L(i, j) = U(i, j);
213       }
214   }
215
216   for (int i = 0; i < rows(A); ++i) {
217       for (int j = 0; j < i; ++j) {
218           U(j, i) = 0;
219       }
220   }
221
222   for (int i = 0; i < rows(A); ++i) {
223       for (int j = 0; j < i; ++j) {
224           L(i, j) = 0;
225       }
226   }
227
228   for (int i = 0; i < rows(A); ++i) {
229       for (int j = 0; j < i; ++j) {
230           L(i, j) = U(i, j);
231       }
232   }
233
234   for (int i = 0; i < rows(A); ++i) {
235       for (int j = 0; j < i; ++j) {
236           U(j, i) = 0;
237       }
238   }
239
240   for (int i = 0; i < rows(A); ++i) {
241       for (int j = 0; j < i; ++j) {
242           L(i, j) = 0;
243       }
244   }
245
246   for (int i = 0; i < rows(A); ++i) {
247       for (int j = 0; j < i; ++j) {
248           L(i, j) = U(i, j);
249       }
250   }
251
252   for (int i = 0; i < rows(A); ++i) {
253       for (int j = 0; j < i; ++j) {
254           U(j, i) = 0;
255       }
256   }
257
258   for (int i = 0; i < rows(A); ++i) {
259       for (int j = 0; j < i; ++j) {
260           L(i, j) = 0;
261       }
262   }
263
264   for (int i = 0; i < rows(A); ++i) {
265       for (int j = 0; j < i; ++j) {
266           L(i, j) = U(i, j);
267       }
268   }
269
270   for (int i = 0; i < rows(A); ++i) {
271       for (int j = 0; j < i; ++j) {
272           U(j, i) = 0;
273       }
274   }
275
276   for (int i = 0; i < rows(A); ++i) {
277       for (int j = 0; j < i; ++j) {
278           L(i, j) = 0;
279       }
280   }
281
282   for (int i = 0; i < rows(A); ++i) {
283       for (int j = 0; j < i; ++j) {
284           L(i, j) = U(i, j);
285       }
286   }
287
288   for (int i = 0; i < rows(A); ++i) {
289       for (int j = 0; j < i; ++j) {
290           U(j, i) = 0;
291       }
292   }
293
294   for (int i = 0; i < rows(A); ++i) {
295       for (int j = 0; j < i; ++j) {
296           L(i, j) = 0;
297       }
298   }
299
300   for (int i = 0; i < rows(A); ++i) {
301       for (int j = 0; j < i; ++j) {
302           L(i, j) = U(i, j);
303       }
304   }
305
306   for (int i = 0; i < rows(A); ++i) {
307       for (int j = 0; j < i; ++j) {
308           U(j, i) = 0;
309       }
310   }
311
312   for (int i = 0; i < rows(A); ++i) {
313       for (int j = 0; j < i; ++j) {
314           L(i, j) = 0;
315       }
316   }
317
318   for (int i = 0; i < rows(A); ++i) {
319       for (int j = 0; j < i; ++j) {
320           L(i, j) = U(i, j);
321       }
322   }
323
324   for (int i = 0; i < rows(A); ++i) {
325       for (int j = 0; j < i; ++j) {
326           U(j, i) = 0;
327       }
328   }
329
330   for (int i = 0; i < rows(A); ++i) {
331       for (int j = 0; j < i; ++j) {
332           L(i, j) = 0;
333       }
334   }
335
336   for (int i = 0; i < rows(A); ++i) {
337       for (int j = 0; j < i; ++j) {
338           L(i, j) = U(i, j);
339       }
340   }
341
342   for (int i = 0; i < rows(A); ++i) {
343       for (int j = 0; j < i; ++j) {
344           U(j, i) = 0;
345       }
346   }
347
348   for (int i = 0; i < rows(A); ++i) {
349       for (int j = 0; j < i; ++j) {
350           L(i, j) = 0;
351       }
352   }
353
354   for (int i = 0; i < rows(A); ++i) {
355       for (int j = 0; j < i; ++j) {
356           L(i, j) = U(i, j);
357       }
358   }
359
360   for (int i = 0; i < rows(A); ++i) {
361       for (int j = 0; j < i; ++j) {
362           U(j, i) = 0;
363       }
364   }
365
366   for (int i = 0; i < rows(A); ++i) {
367       for (int j = 0; j < i; ++j) {
368           L(i, j) = 0;
369       }
370   }
371
372   for (int i = 0; i < rows(A); ++i) {
373       for (int j = 0; j < i; ++j) {
374           L(i, j) = U(i, j);
375       }
376   }
377
378   for (int i = 0; i < rows(A); ++i) {
379       for (int j = 0; j < i; ++j) {
380           U(j, i) = 0;
381       }
382   }
383
384   for (int i = 0; i < rows(A); ++i) {
385       for (int j = 0; j < i; ++j) {
386           L(i, j) = 0;
387       }
388   }
389
390   for (int i = 0; i < rows(A); ++i) {
391       for (int j = 0; j < i; ++j) {
392           L(i, j) = U(i, j);
393       }
394   }
395
396   for (int i = 0; i < rows(A); ++i) {
397       for (int j = 0; j < i; ++j) {
398           U(j, i) = 0;
399       }
400   }
401
402   for (int i = 0; i < rows(A); ++i) {
403       for (int j = 0; j < i; ++j) {
404           L(i, j) = 0;
405       }
406   }
407
408   for (int i = 0; i < rows(A); ++i) {
409       for (int j = 0; j < i; ++j) {
410           L(i, j) = U(i, j);
411       }
412   }
413
414   for (int i = 0; i < rows(A); ++i) {
415       for (int j = 0; j < i; ++j) {
416           U(j, i) = 0;
417       }
418   }
419
420   for (int i = 0; i < rows(A); ++i) {
421       for (int j = 0; j < i; ++j) {
422           L(i, j) = 0;
423       }
424   }
425
426   for (int i = 0; i < rows(A); ++i) {
427       for (int j = 0; j < i; ++j) {
428           L(i, j) = U(i, j);
429       }
430   }
431
432   for (int i = 0; i < rows(A); ++i) {
433       for (int j = 0; j < i; ++j) {
434           U(j, i) = 0;
435       }
436   }
437
438   for (int i = 0; i < rows(A); ++i) {
439       for (int j = 0; j < i; ++j) {
440           L(i, j) = 0;
441       }
442   }
443
444   for (int i = 0; i < rows(A); ++i) {
445       for (int j = 0; j < i; ++j) {
446           L(i, j) = U(i, j);
447       }
448   }
449
450   for (int i = 0; i < rows(A); ++i) {
451       for (int j = 0; j < i; ++j) {
452           U(j, i) = 0;
453       }
454   }
455
456   for (int i = 0; i < rows(A); ++i) {
457       for (int j = 0; j < i; ++j) {
458           L(i, j) = 0;
459       }
460   }
461
462   for (int i = 0; i < rows(A); ++i) {
463       for (int j = 0; j < i; ++j) {
464           L(i, j) = U(i, j);
465       }
466   }
467
468   for (int i = 0; i < rows(A); ++i) {
469       for (int j = 0; j < i; ++j) {
470           U(j, i) = 0;
471       }
472   }
473
474   for (int i = 0; i < rows(A); ++i) {
475       for (int j = 0; j < i; ++j) {
476           L(i, j) = 0;
477       }
478   }
479
480   for (int i = 0; i < rows(A); ++i) {
481       for (int j = 0; j < i; ++j) {
482           L(i, j) = U(i, j);
483       }
484   }
485
486   for (int i = 0; i < rows(A); ++i) {
487       for (int j = 0; j < i; ++j) {
488           U(j, i) = 0;
489       }
490   }
491
492   for (int i = 0; i < rows(A); ++i) {
493       for (int j = 0; j < i; ++j) {
494           L(i, j) = 0;
495       }
496   }
497
498   for (int i = 0; i < rows(A); ++i) {
499       for (int j = 0; j < i; ++j) {
500           L(i, j) = U(i, j);
501       }
502   }
503
504   for (int i = 0; i < rows(A); ++i) {
505       for (int j = 0; j < i; ++j) {
506           U(j, i) = 0;
507       }
508   }
509
510   for (int i = 0; i < rows(A); ++i) {
511       for (int j = 0; j < i; ++j) {
512           L(i, j) = 0;
513       }
514   }
515
516   for (int i = 0; i < rows(A); ++i) {
517       for (int j = 0; j < i; ++j) {
518           L(i, j) = U(i, j);
519       }
520   }
521
522   for (int i = 0; i < rows(A); ++i) {
523       for (int j = 0; j < i; ++j) {
524           U(j, i) = 0;
525       }
526   }
527
528   for (int i = 0; i < rows(A); ++i) {
529       for (int j = 0; j < i; ++j) {
530           L(i, j) = 0;
531       }
532   }
533
534   for (int i = 0; i < rows(A); ++i) {
535       for (int j = 0; j < i; ++j) {
536           L(i, j) = U(i, j);
537       }
538   }
539
540   for (int i = 0; i < rows(A); ++i) {
541       for (int j = 0; j < i; ++j) {
542           U(j, i) = 0;
543       }
544   }
545
546   for (int i = 0; i < rows(A); ++i) {
547       for (int j = 0; j < i; ++j) {
548           L(i, j) = 0;
549       }
550   }
551
552   for (int i = 0; i < rows(A); ++i) {
553       for (int j = 0; j < i; ++j) {
554           L(i, j) = U(i, j);
555       }
556   }
557
558   for (int i = 0; i < rows(A); ++i) {
559       for (int j = 0; j < i; ++j) {
560           U(j, i) = 0;
561       }
562   }
563
564   for (int i = 0; i < rows(A); ++i) {
565       for (int j = 0; j < i; ++j) {
566           L(i, j) = 0;
567       }
568   }
569
570   for (int i = 0; i < rows(A); ++i) {
571       for (int j = 0; j < i; ++j) {
572           L(i, j) = U(i, j);
573       }
574   }
575
576   for (int i = 0; i < rows(A); ++i) {
577       for (int j = 0; j < i; ++j) {
578           U(j, i) = 0;
579       }
580   }
581
582   for (int i = 0; i < rows(A); ++i) {
583       for (int j = 0; j < i; ++j) {
584           L(i, j) = 0;
585       }
586   }
587
588   for (int i = 0; i < rows(A); ++i) {
589       for (int j = 0; j < i; ++j) {
590           L(i, j) = U(i, j);
591       }
592   }
593
594   for (int i = 0; i < rows(A); ++i) {
595       for (int j = 0; j < i; ++j) {
596           U(j, i) = 0;
597       }
598   }
599
600   for (int i = 0; i < rows(A); ++i) {
601       for (int j = 0; j < i; ++j) {
602           L(i, j) = 0;
603       }
604   }
605
606   for (int i = 0; i < rows(A); ++i) {
607       for (int j = 0; j < i; ++j) {
608           L(i, j) = U(i, j);
609       }
610   }
611
612   for (int i = 0; i < rows(A); ++i) {
613       for (int j = 0; j < i; ++j) {
614           U(j, i) = 0;
615       }
616   }
617
618   for (int i = 0; i < rows(A); ++i) {
619       for (int j = 0; j < i; ++j) {
620           L(i, j) = 0;
621       }
622   }
623
624   for (int i = 0; i < rows(A); ++i) {
625       for (int j = 0; j < i; ++j) {
626           L(i, j) = U(i, j);
627       }
628   }
629
630   for (int i = 0; i < rows(A); ++i) {
631       for (int j = 0; j < i; ++j) {
632           U(j, i) = 0;
633       }
634   }
635
636   for (int i = 0; i < rows(A); ++i) {
637       for (int j = 0; j < i; ++j) {
638           L(i, j) = 0;
639       }
640   }
641
642   for (int i = 0; i < rows(A); ++i) {
643       for (int j = 0; j < i; ++j) {
644           L(i, j) = U(i, j);
645       }
646   }
647
648   for (int i = 0; i < rows(A); ++i) {
649       for (int j = 0; j < i; ++j) {
650           U(j, i) = 0;
651       }
652   }
653
654   for (int i = 0; i < rows(A); ++i) {
655       for (int j = 0; j < i; ++j) {
656           L(i, j) = 0;
657       }
658   }
659
660   for (int i = 0; i < rows(A); ++i) {
661       for (int j = 0; j < i; ++j) {
662           L(i, j) = U(i, j);
663       }
664   }
665
666   for (int i = 0; i < rows(A); ++i) {
667       for (int j = 0; j < i; ++j) {
668           U(j, i) = 0;
669       }
670   }
671
672   for (int i = 0; i < rows(A); ++i) {
673       for (int j = 0; j < i; ++j) {
674           L(i, j) = 0;
675       }
676   }
677
678   for (int i = 0; i < rows(A); ++i) {
679       for (int j = 0; j < i; ++j) {
680           L(i, j) = U(i, j);
681       }
682   }
683
684   for (int i = 0; i < rows(A); ++i) {
685       for (int j = 0; j < i; ++j) {
686           U(j, i) = 0;
687       }
688   }
689
690   for (int i = 0; i < rows(A); ++i) {
691       for (int j = 0; j < i; ++j) {
692           L(i, j) = 0;
693       }
694   }
695
696   for (int i = 0; i < rows(A); ++i) {
697       for (int j = 0; j < i; ++j) {
698           L(i, j) = U(i, j);
699       }
700   }
701
702   for (int i = 0; i < rows(A); ++i) {
703       for (int j = 0; j < i; ++j) {
704           U(j, i) = 0;
705       }
706   }
707
708   for (int i = 0; i < rows(A); ++i) {
709       for (int j = 0; j < i; ++j) {
710           L(i, j) = 0;
711       }
712   }
713
714   for (int i = 0; i < rows(A); ++i) {
715       for (int j = 0; j < i; ++j) {
716           L(i, j) = U(i, j);
717       }
718   }
719
720   for (int i = 0; i < rows(A); ++i) {
721       for (int j = 0; j < i; ++j) {
722           U(j, i) = 0;
723       }
724   }
725
726   for (int i = 0; i < rows(A); ++i) {
727       for (int j = 0; j < i; ++j) {
728           L(i, j) = 0;
729       }
730   }
731
732   for (int i = 0; i < rows(A); ++i) {
733       for (int j = 0; j < i; ++j) {
734           L(i, j) = U(i, j);
735       }
736   }
737
738   for (int i = 0; i < rows(A); ++i) {
739       for (int j = 0; j < i; ++j) {
740           U(j, i) = 0;
741       }
742   }
743
744   for (int i = 0; i < rows(A); ++i) {
745       for (int j = 0; j < i; ++j) {
746           L(i, j) = 0;
747       }
748   }
749
750   for (int i = 0; i < rows(A); ++i) {
751       for (int j = 0; j < i; ++j) {
752           L(i, j) = U(i, j);
753       }
754   }
755
756   for (int i = 0; i < rows(A); ++i) {
757       for (int j = 0; j < i; ++j) {
758           U(j, i) = 0;
759       }
760   }
761
762   for (int i = 0; i < rows(A); ++i) {
763       for (int j = 0; j < i; ++j) {
764           L(i, j) = 0;
765       }
766   }
767
768   for (int i = 0; i < rows(A); ++i) {
769       for (int j = 0; j < i; ++j) {
770           L(i, j) = U(i, j);
771       }
772   }
773
774   for (int i = 0; i < rows(A); ++i) {
775       for (int j = 0; j < i; ++j) {
776           U(j, i) = 0;
777       }
778   }
779
780   for (int i = 0; i < rows(A); ++i) {
781       for (int j = 0; j < i; ++j) {
782           L(i, j) = 0;
783       }
784   }
785
786   for (int i = 0; i < rows(A); ++i) {
787       for (int j = 0; j < i; ++j) {
788           L(i, j) = U(i, j);
789       }
790   }
791
792   for (int i = 0; i < rows(A); ++i) {
793       for (int j = 0; j < i; ++j) {
794           U(j, i) = 0;
795       }
796   }
797
798   for (int i = 0; i < rows(A); ++i) {
799       for (int j = 0; j < i; ++j) {
800           L(i, j) = 0;
801       }
802   }
803
804   for (int i = 0; i < rows(A); ++i) {
805       for (int j = 0; j < i; ++j) {
806           L(i, j) = U(i, j);
807       }
808   }
809
810   for (int i = 0; i < rows(A); ++i) {
811       for (int j = 0; j < i; ++j) {
812           U(j, i) = 0;
813       }
814   }
815
816   for (int i = 0; i < rows(A); ++i) {
817       for (int j = 0; j < i; ++j) {
818           L(i, j) = 0;
819       }
820   }
821
822   for (int i = 0; i < rows(A); ++i) {
823       for (int j = 0; j < i; ++j) {
824           L(i, j) = U(i, j);
825       }
826   }
827
828   for (int i = 0; i < rows(A); ++i) {
829       for (int j = 0; j < i; ++j) {
830           U(j,
```

```

4 }
5
6 if (rows(A) % 2 == 0) {
7     memCounterEnterCall(rows(A), cols(A), 4);
8
9     int halfSize = rows(A) / 2;
10
11    Matrix A11 = subMatrix(A, 0, 0, halfSize, halfSize);
12    Matrix A12 = subMatrix(A, 0, halfSize, halfSize, halfSize)
13        ;
14    Matrix A21 = subMatrix(A, halfSize, 0, halfSize, halfSize)
15        ;
16    Matrix A22 = subMatrix(A, halfSize, halfSize, halfSize,
17        halfSize);
18
19    Matrix b1 = subMatrix(b, 0, 0, halfSize, 1);
20    Matrix b2 = subMatrix(b, halfSize, 0, halfSize, 1);
21
22    auto [L11, U11] = LUfactorization(A11, multImpl);
23
24    Matrix L11_inv = inverse(L11, multImpl);
25    Matrix U11_inv = inverse(U11, multImpl);
26
27    Matrix S1 = multImpl->multiply(A21, U11_inv);
28    Matrix S2 = multImpl->multiply(L11_inv, A12);
29    Matrix S3 = L11_inv * b1;
30
31    auto [LS, US] = LUfactorization(A22 - multImpl->multiply(
32        S1, S2), multImpl);
33
34    Matrix LS_inv = inverse(LS, multImpl);
35
36    Matrix c1 = S3;
37    Matrix c2 = LS_inv * b2 - multImpl->multiply(LS_inv, S1) *
38        S3;
39
40    Matrix C11 = U11;
41    Matrix C12 = multImpl->multiply(L11, A12);
42    Matrix C21 = zeroMatrix(halfSize, halfSize);
43    Matrix C22 = US;
44
45    Matrix C = combine(C11, C12,
46                        C21, C22);
47    Matrix c = combine(c1, {},
48                        c2, {});
49
50    memCounterExitCall(rows(A), cols(A), 4);
51    return {C, c};
52} else {

```

```

48     Matrix A_padded = pad(A, rows(A) + 1, rows(A) + 1);
49     Matrix b_padded = pad(b, rows(b) + 1, 1);
50     A_padded[rows(A)][rows(A)] = 1.0;
51     b_padded[rows(b)][0] = 0.0;
52     auto [C_padded, c_padded] = GaussElimination(A_padded,
53           b_padded, multImpl);
54     Matrix C = trim(C_padded, rows(A), rows(A));
55     Matrix c = trim(c_padded, rows(b), 1);
56     return {C, c};
57 }
}

```

2.2 Wykresy

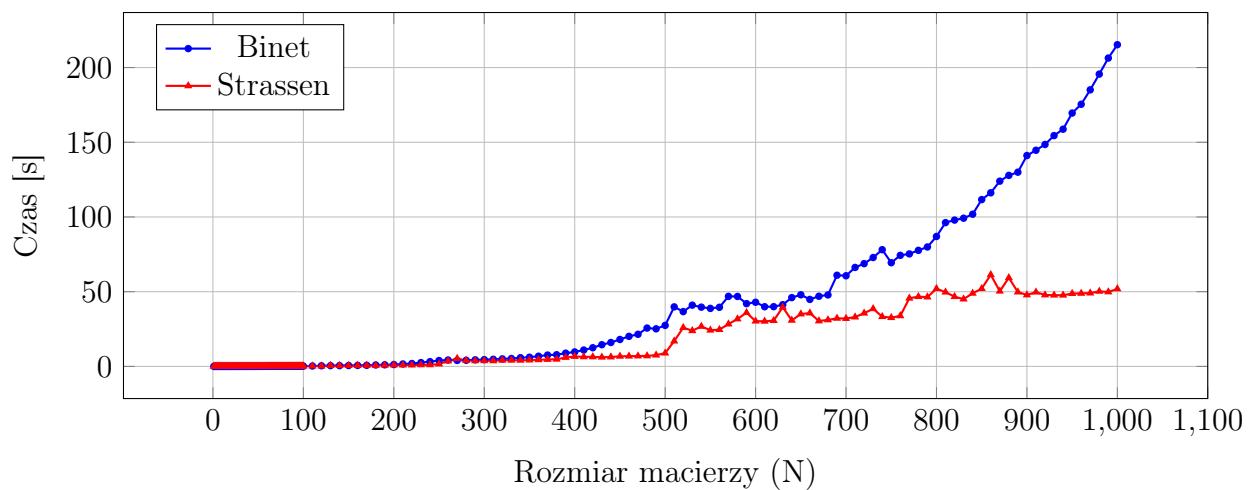


Figure 2.2.1: Czas działania (Binet vs Strassen)

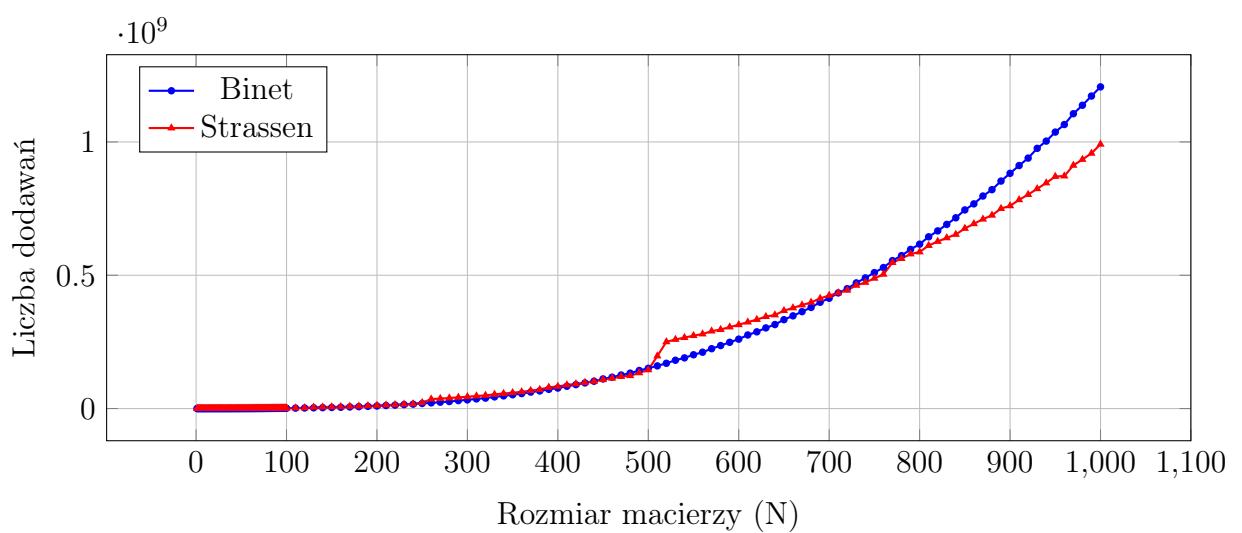


Figure 2.2.2: Porównanie liczby operacji dodawania

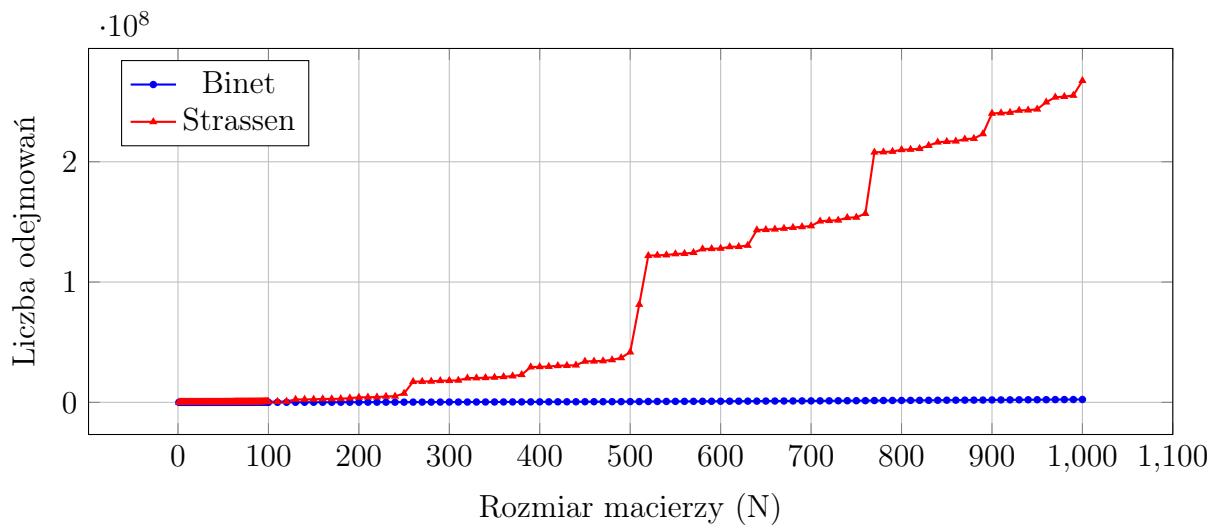


Figure 2.2.3: Porównanie liczby operacji odejmowania

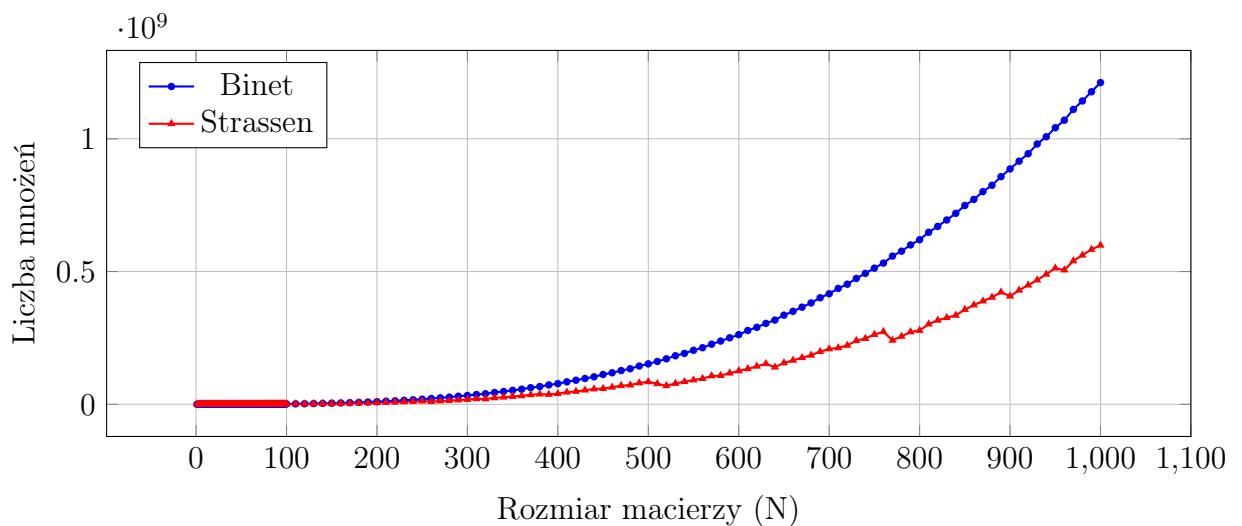


Figure 2.2.4: Porównanie liczby operacji mnożenia

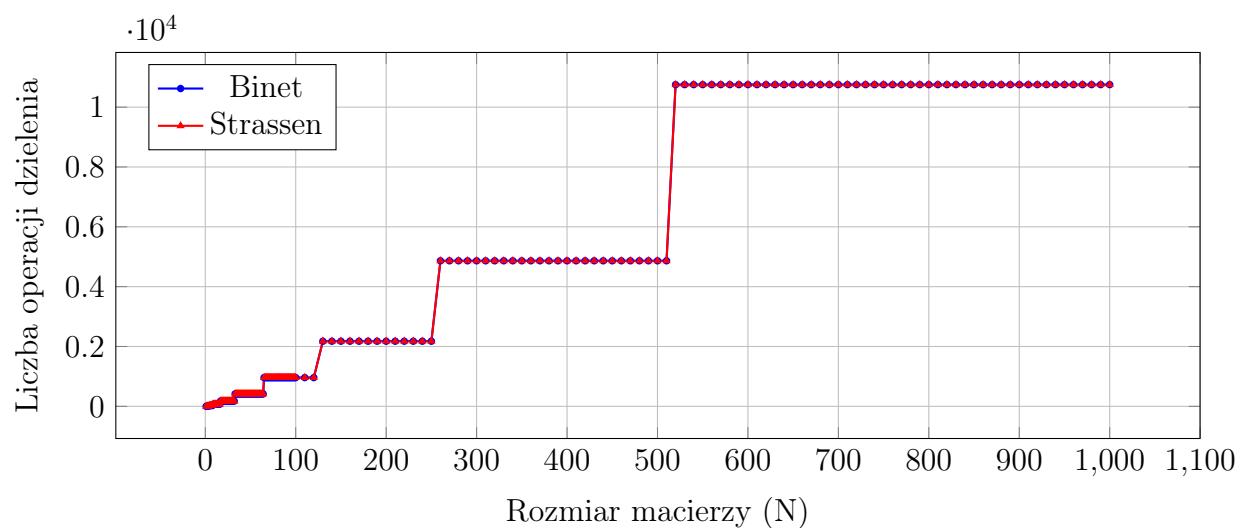


Figure 2.2.5: Porównanie liczby operacji dzielenia

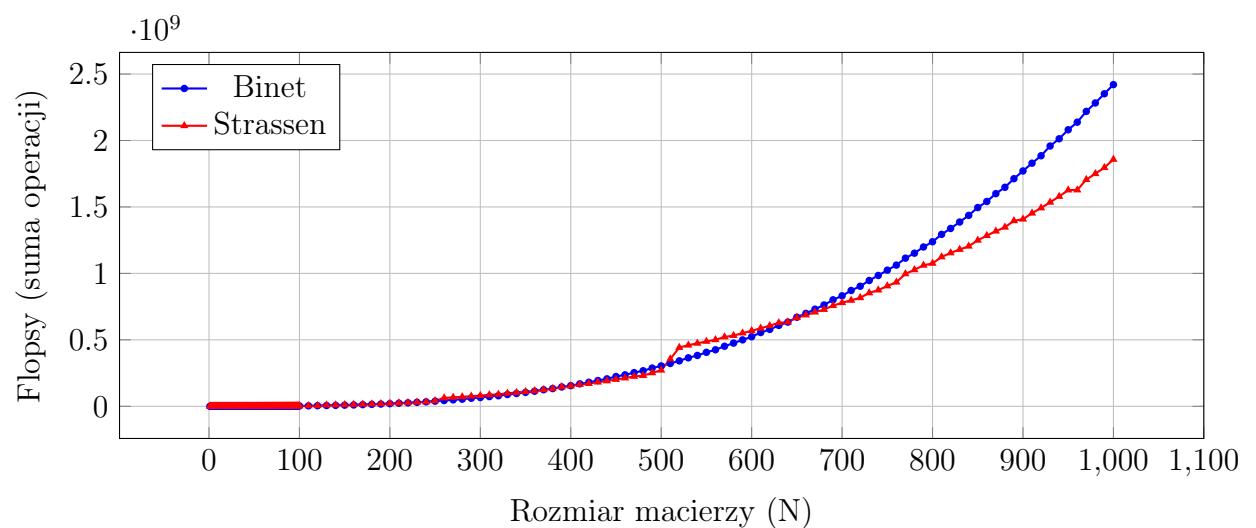


Figure 2.2.6: Porównanie liczby operacji zmiennoprzecinkowych (flops)

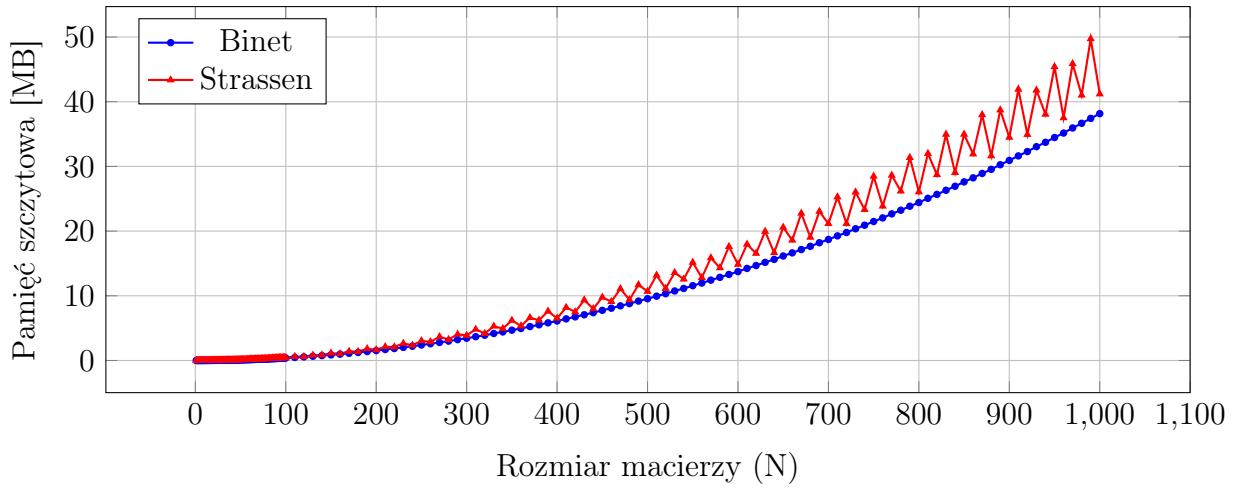


Figure 2.2.7: Porównanie zużycia pamięci

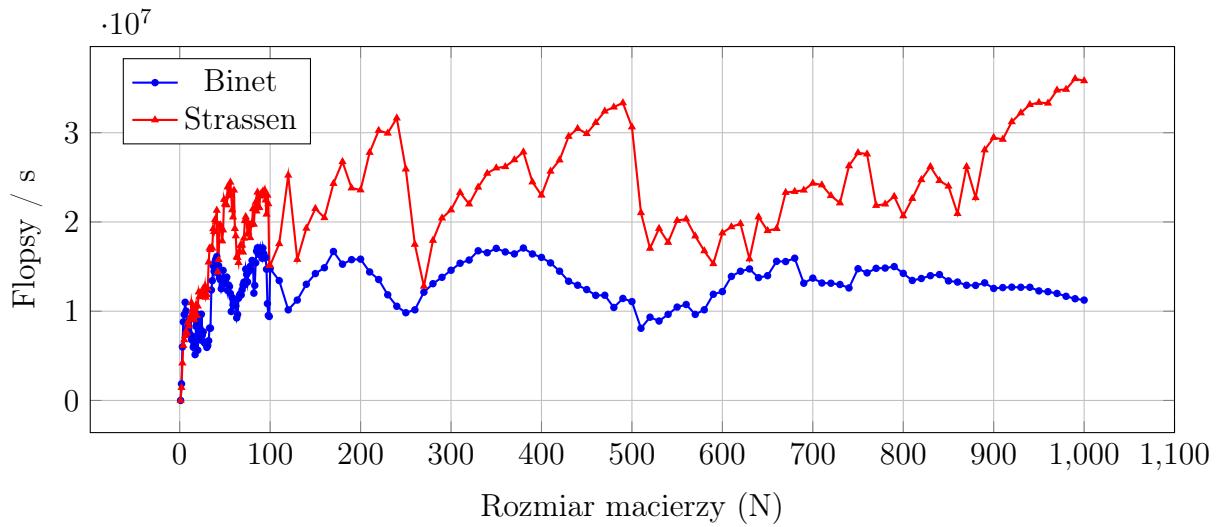


Figure 2.2.8: Przepustowość (flops / s) porównanie

3 LU faktoryzacja

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad (12)$$

$$L_{11}U_{11} = LU(A_{11}) \quad (13)$$

$$S = A_{22} - A_{21}U_{11}^{-1}L_{11}^{-1}A_{12} = L_S U_S \quad (14)$$

$$LU(A) = \begin{bmatrix} L_{11} & 0 \\ A_{21}U_{11}^{-1} & L_S \end{bmatrix} \begin{bmatrix} U_{11} & L_{11}^{-1}A_{12} \\ 0 & U_S \end{bmatrix} \quad (15)$$

Wyznacznik $A = LU$ to $\det(A) = \prod_1^n u_{nn}$.

3.1 Implementacja

```
1 std::pair<Matrix, Matrix> LUfactorization(const Matrix &A, std::unique_ptr<IMnozenie> &multImpl) {
2     if (rows(A) == 1) {
3         Matrix L = identityMatrix(1);
4         Matrix U = A;
5         return {L, U};
6     }
7
8     if (rows(A) % 2 == 0) {
9         memCounterEnterCall(rows(A), cols(A), 3);
10
11     int halfSize = rows(A) / 2;
12
13     Matrix A11 = subMatrix(A, 0, 0, halfSize, halfSize);
14     Matrix A12 = subMatrix(A, 0, halfSize, halfSize, halfSize)
15         ;
16     Matrix A21 = subMatrix(A, halfSize, 0, halfSize, halfSize)
17         ;
18     Matrix A22 = subMatrix(A, halfSize, halfSize, halfSize,
19         halfSize);
20
21     auto [L11, U11] = LUfactorization(A11, multImpl);
22
23     Matrix L11_inv = inverse(L11, multImpl);
24     Matrix U11_inv = inverse(U11, multImpl);
25
26     Matrix U12 = multImpl->multiply(L11_inv, A12);
27     Matrix L21 = multImpl->multiply(A21, U11_inv);
28
29     Matrix S = A22 - multImpl->multiply(L21, U12);
30
31     auto [L22, U22] = LUfactorization(S, multImpl);
32
33     Matrix L = combine(L11, zeroMatrix(halfSize, halfSize),
34                         L21, L22);
35     Matrix U = combine(U11, U12,
36                         zeroMatrix(halfSize, halfSize), U22);
37
38     memCounterExitCall(rows(A), cols(A), 3);
39     return {L, U};
40 } else {
41     Matrix A_padded = pad(A, rows(A) + 1, rows(A) + 1);
42     auto [L_padded, U_padded] = LUfactorization(A_padded,
43         multImpl);
44     Matrix L = trim(L_padded, rows(A), rows(A));
45     Matrix U = trim(U_padded, rows(A), rows(A));
46     return {L, U};
```

```

43     }
44 }
45
46 double determinantLU(const Matrix &A, std::unique_ptr<IMnozenie> &
47   multImpl) {
48   auto [_, U] = LUfactorization(A, multImpl);
49   double det = 1.0;
50   for (int i = 0; i < rows(U); ++i) {
51     det *= U[i][i];
52   }
53   return det;
}

```

3.2 Wykresy

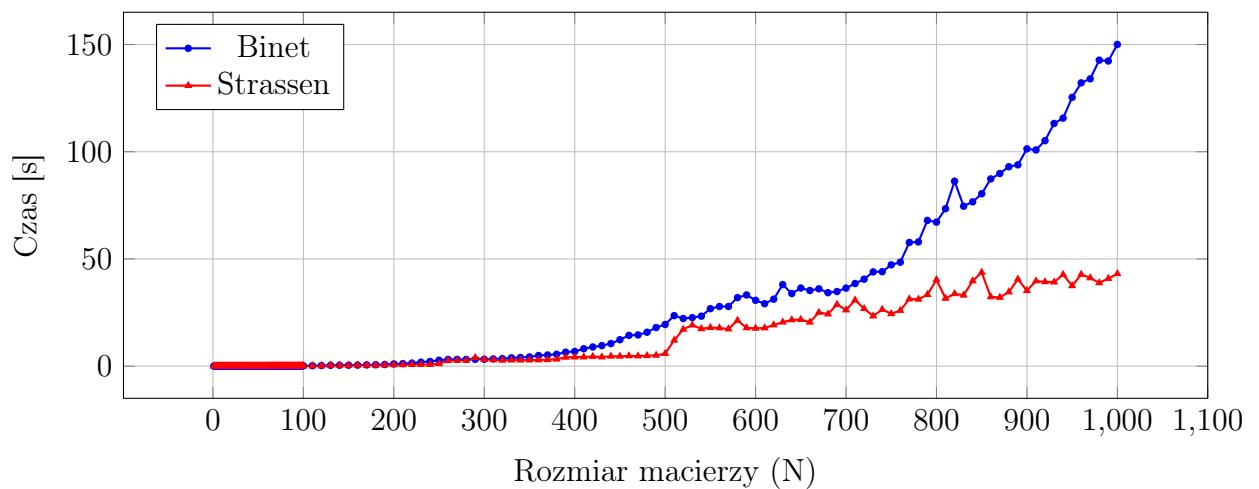


Figure 3.2.1: Czas działania (Binet vs Strassen)

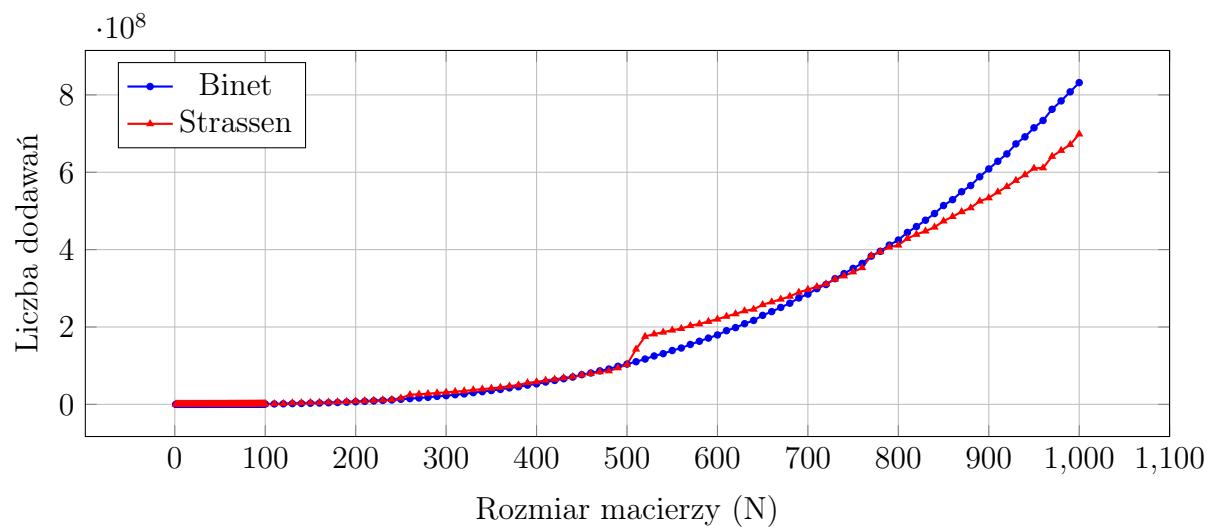


Figure 3.2.2: Porównanie liczby operacji dodawania

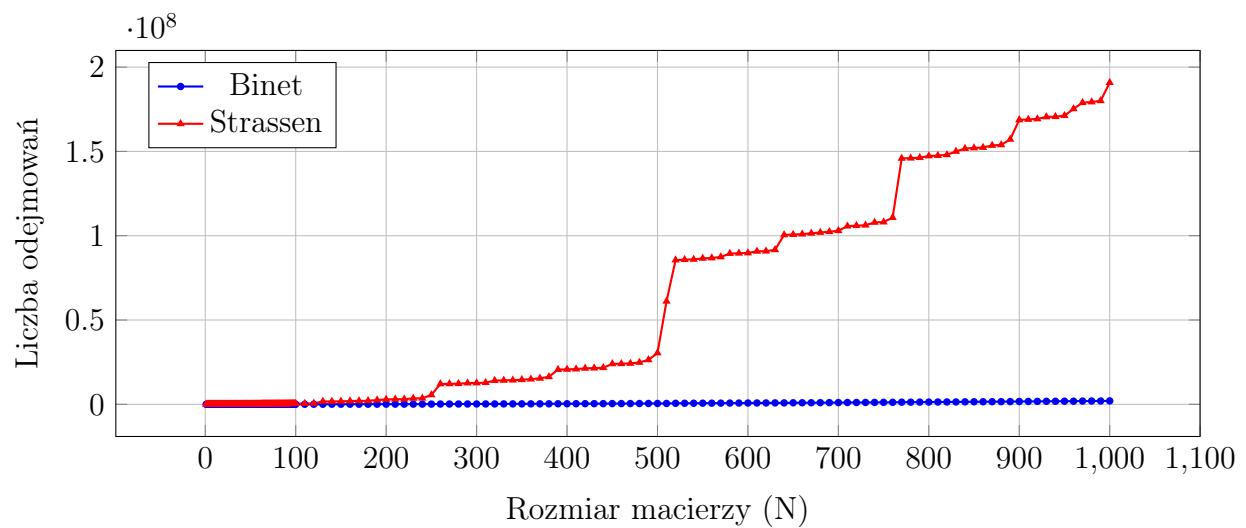


Figure 3.2.3: Porównanie liczby operacji odejmowania

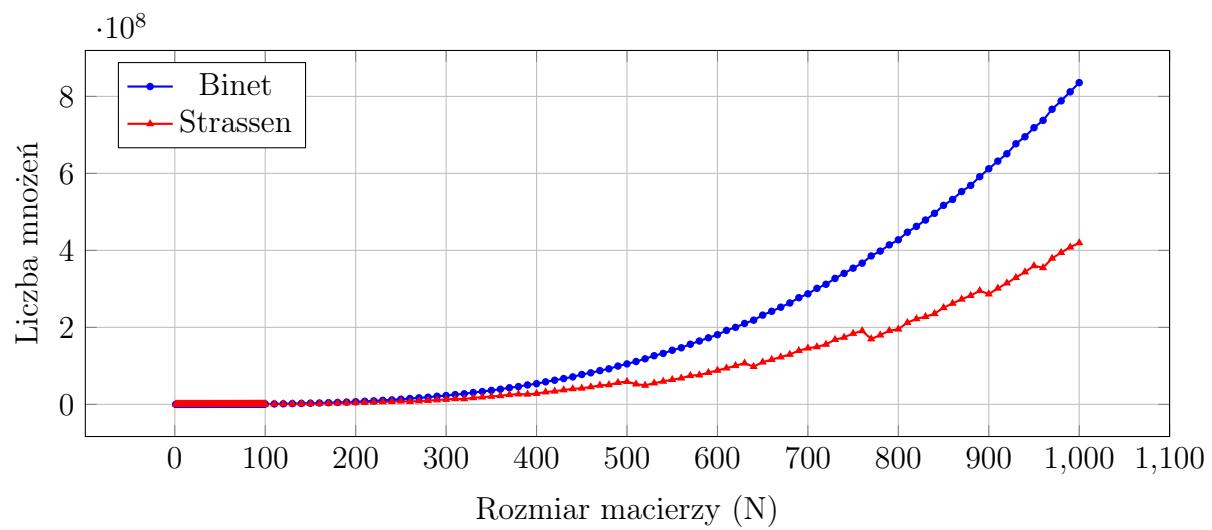


Figure 3.2.4: Porównanie liczby operacji mnożenia

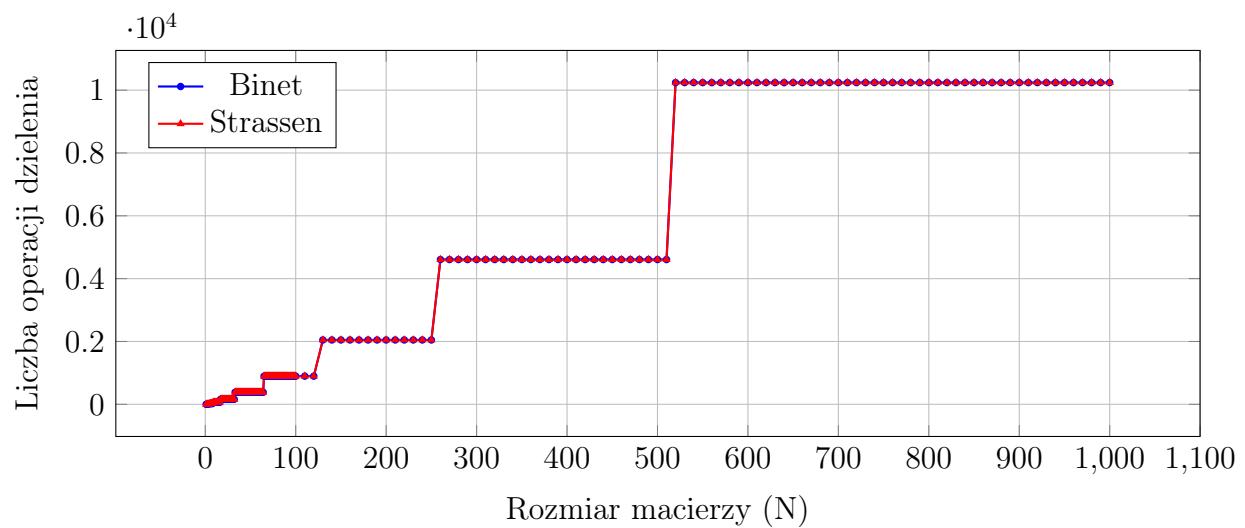


Figure 3.2.5: Porównanie liczby operacji dzielenia

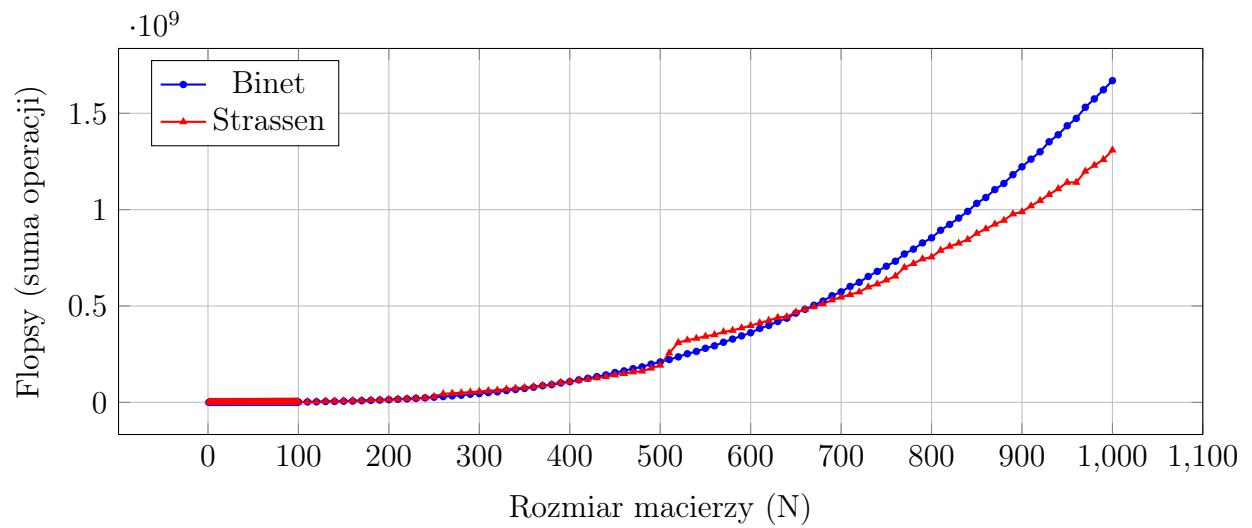


Figure 3.2.6: Porównanie liczby operacji zmiennoprzecinkowych (flops)

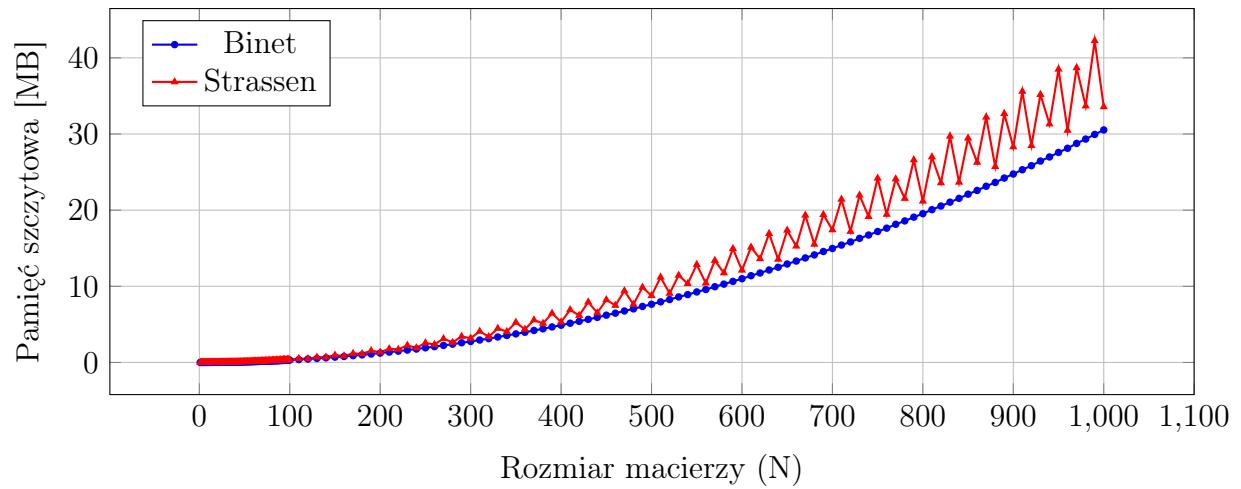


Figure 3.2.7: Porównanie zużycia pamięci

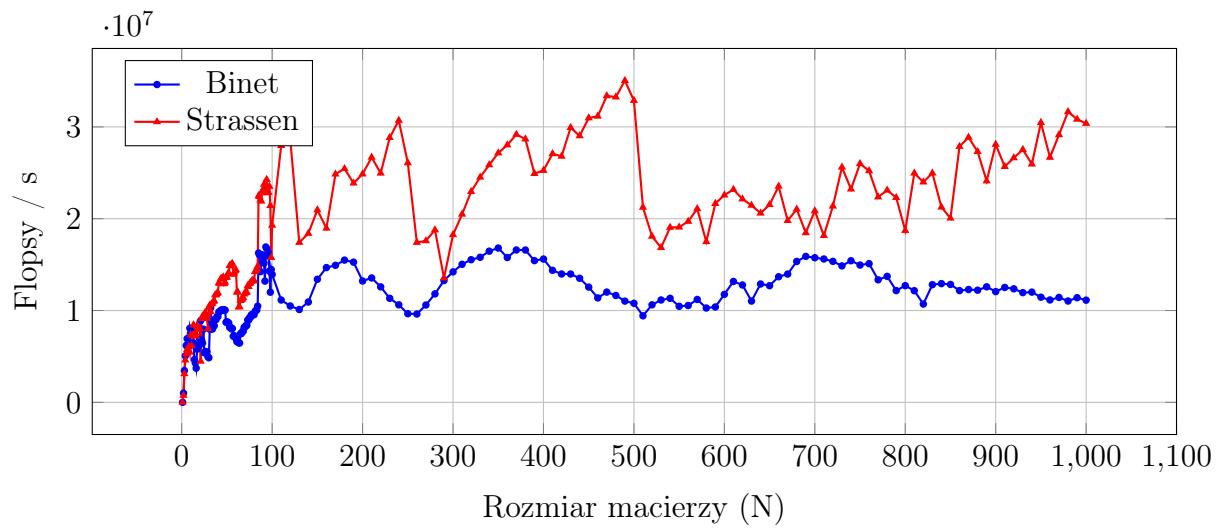


Figure 3.2.8: Przepustowość (flops / s) porównanie