

C と C++ の並列計算パフォーマンス

Parallel Computing Performance of C and C++

早稲田大学大学院 基幹理工学研究科

情報理工・情報通信専攻

上田研究室 修士課程学生

5121F099 鍾中

1. 先行研究

2. C の並列計算パフォーマンス

3. C++ の並列計算パフォーマンス

4. C と C++ のパフォーマンス比較

5. Nested Pointer 配列について

6. 参考文献

先行研究

Previous Research

- キャッシュの Hit Rate と Miss Rate:
 - Hit Rate はある時間間隔以内のキャッシュヒットの数をメモリ要求の総数で割った値。
 - Miss Rate は 100% マイナス Hit Rate。

$$hit\ rate = \left(\frac{cache\ hits}{memory\ request} \right) \times 100\%$$

$$miss\ rate = 100\% - hit\ rate$$

先行研究

Previous Research

- 行列の掛け算の方法が異なれば Miss Rate も異なる：
 - kij あるいは ikj の Miss Rate が一番小さい
 - ➡ 性能が一番いい
 - jki あるいは kji の Miss Rate が一番大きい
 - ➡ 性能が一番悪い

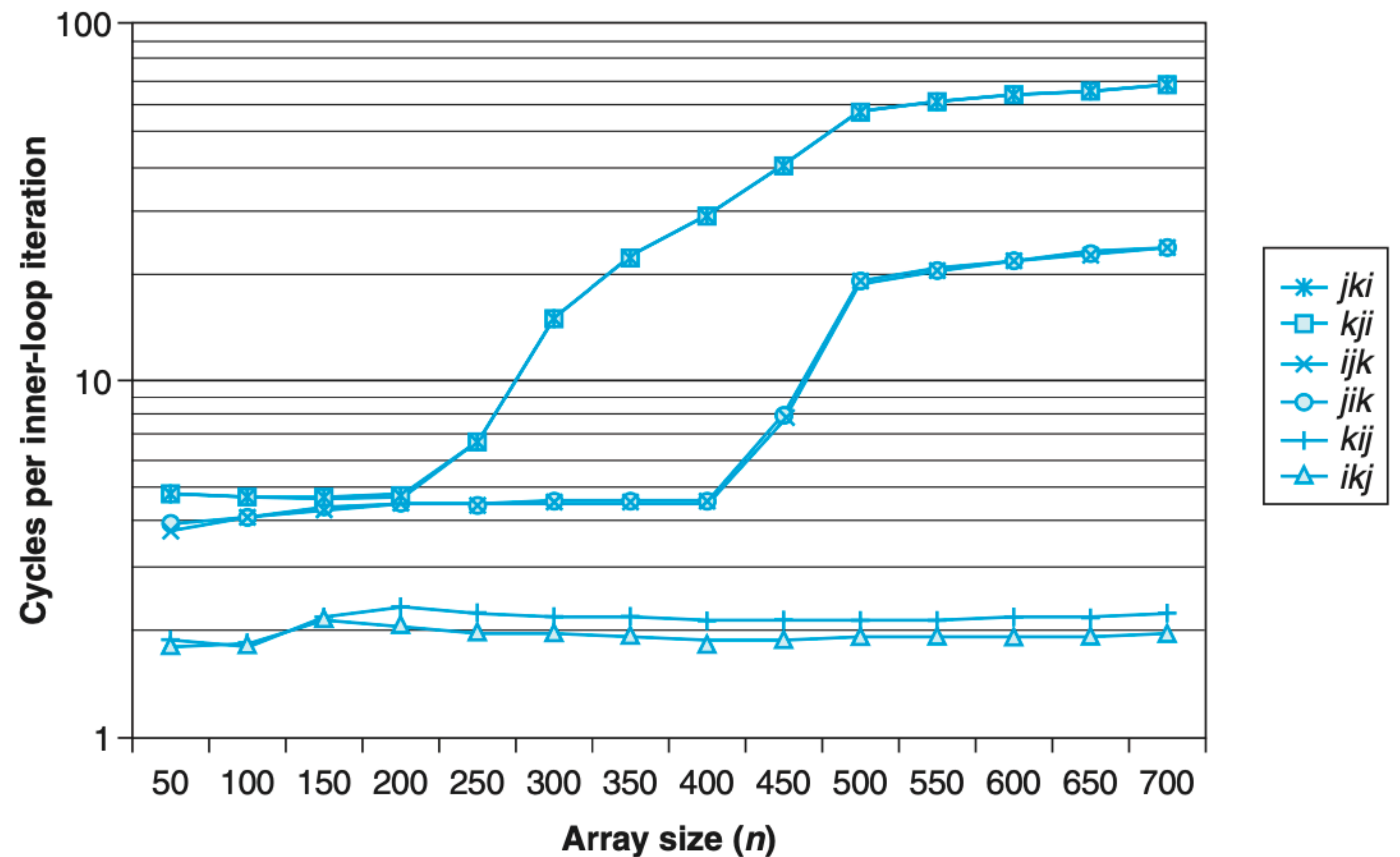
先行研究

Previous Research

行列の掛け算の性能

(Core i7 System)

**CS : APP3e* §6.6.2



1. 先行研究

2. C の並列計算パフォーマンス

3. C++ の並列計算パフォーマンス

4. C と C++ のパフォーマンス比較

5. Nested Pointer 配列について

6. 参考文献

C の並列計算パフォーマンス

Parallel Computing Performance of C

- C の行列掛け算プログラムは pthreads を使って並列化した
- 行列のデータ構造:

```
int **mat;  
mat = malloc(sizeof(int *) * size);  
for (int i = 0; i < size; ++i) {  
    mat[i] = malloc(sizeof(int) * size);  
}  
mat[i][j]; // 行iのj番目の要素
```

C の並列計算パフォーマンス

Parallel Computing Performance of C

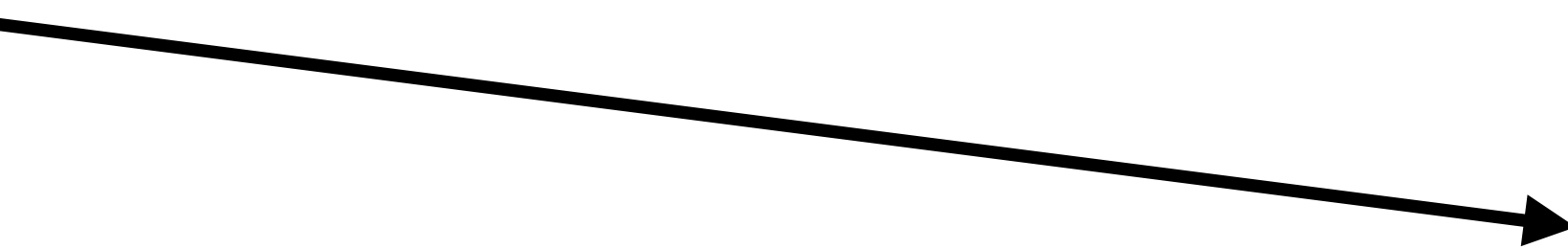
- 行列掛け算の並列化:

```
int block_size = (size * size) / threads_num;
pthread_t threads[threads_num];
struct mmul_t mmul[threads_num];
for (int i = 0; i < threads_num; i++) {
    mmul[i].a = a;
    mmul[i].b = b;
    mmul[i].r = r;
    mmul[i].count = i == threads_num - 1 ? (size * size - block_size * i) : block_size;
    mmul[i].start_row = block_size * i / size;
    mmul[i].start_column = block_size * i % size;
    pthread_create(&threads[i], NULL, mat_mul_thread, (mmul + i));
}
for (int i = 0; i < threads_num; i++)
    pthread_join(threads[i], NULL);
```


C の並列計算パフォーマンス

Parallel Computing Performance of C

```
void *mat_mul_thread(void *mmul_arg) {  
    struct mmul_t *mmul = mmul_arg;  
    int size = mmul->a->size;  
    int k = mmul->start_column;  
    for (int i = mmul->start_row; i < size; i++) {  
        for (; k < size; k++) {  
            int r = mmul->a->matrix[i][k];  
            for (int j = 0; j < size; j++)  
                mmul->r->matrix[i][j] += r * mmul->b->matrix[k][j];  
            if (!(--(mmul->count)))  
                pthread_exit(NULL);  
        }  
        k = 0;  
    }  
    pthread_exit(NULL);  
}
```



```
struct mmul_t {  
    struct matrix *a, *b, *r;  
    unsigned int count;  
    int start_row, start_column;  
};
```

C の並列計算パフォーマンス

Parallel Computing Performance of C

- ベンチマーク： (parmigiano, gcc with option -O3)

threads	1000	2000	3000	4000	5000
1	0.2150945	3.538020625	12.484331125	29.515970375	57.219937875
2	0.195584875	1.7083235	5.059017125	11.6600465	23.125592375
4	0.13335225	1.09289775	3.20339975	7.23954425	14.4661055
8	0.0746825	0.7194285	2.236958	5.19659625	11.09609775
16	0.040905375	0.396547625	1.1564415	3.090512	6.399203125
32	0.026092	0.21728075	0.675382875	1.641752625	3.83599275

C の並列計算パフォーマンス

Parallel Computing Performance of C

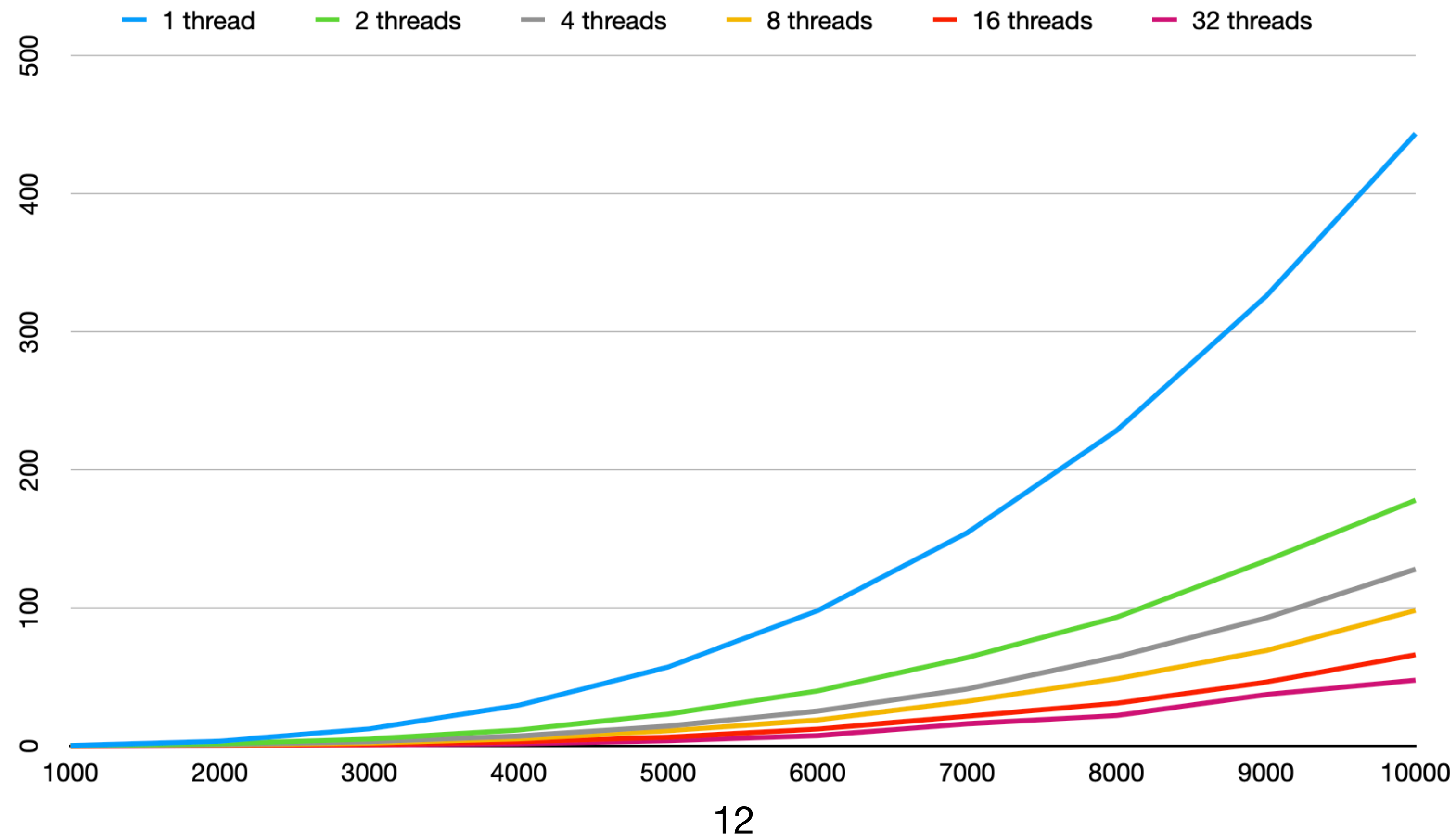
- ベンチマーク： (parmigiano, gcc with option -O3)

threads	6000	7000	8000	9000	10000
1	98.04810162	154.3695232	228.41817987	325.65527625	443.25499037
2	39.91640537	64.0000874	93.12387025	134.20659612	177.91436462
4	25.2697855	41.32330662	64.527921125	92.67895875	128.00913537
8	18.82925862	32.4087495	48.6970355	69.082239	98.189590625
16	12.32761862	21.57344387	30.955059125	46.2589625	66.0418885
32	7.577224125	16.11626737	22.076997	37.251913625	47.655076375

C の並列計算パフォーマンス

Parallel Computing Performance of C

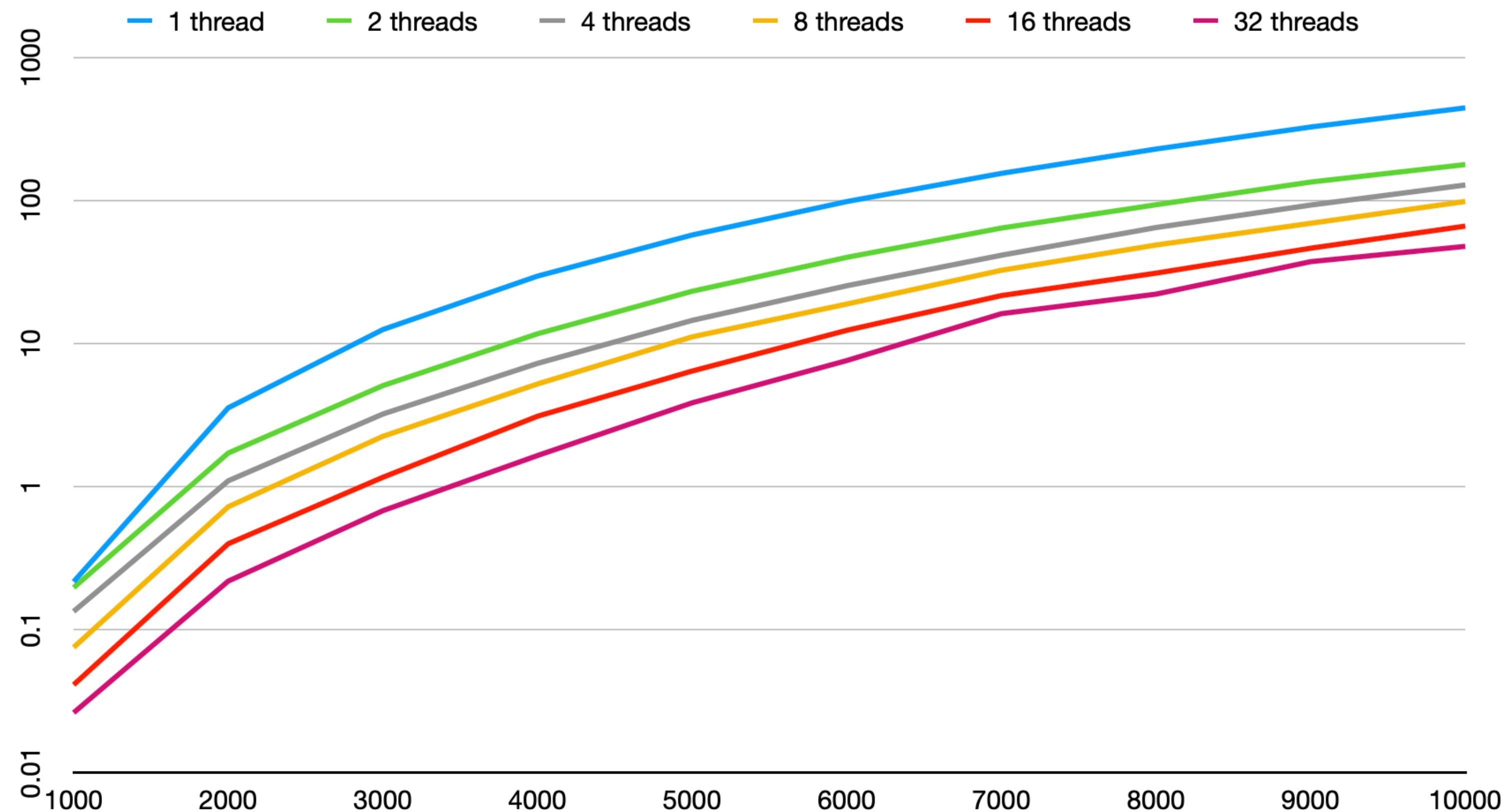
- グラフ:



C の並列計算パフォーマンス

Parallel Computing Performance of C

- グラフ: (Y 軸は対数になる)



1. 先行研究

2. C の並列計算パフォーマンス

3. C++ の並列計算パフォーマンス

4. C と C++ のパフォーマンス比較

5. Nested Pointer 配列について

6. 参考文献

C++ の並列計算パフォーマンス

Parallel Computing Performance of C++

- C++ の行列掛け算プログラムは `std::thread` を使って並列化した
 - `std::thread` ライブラリは、`pthread` に基づき実装される
- データ構造:

```
typedef std::vector<std::vector<int>> matrix;  
matrix mat(size, std::vector<int>(size, 0));  
mat[i][j]; // 行iのj番目の要素
```

C++ の並列計算パフォーマンス

Parallel Computing Performance of C++

- 行列掛け算の並列化:

```
unsigned int block_size = (size * size) / thread_num;
std::vector<std::thread> threads(n: thread_num);
for (int t = 0; t < thread_num; ++t) {
    unsigned int count = t == thread_num - 1 ? (size * size - block_size * t) : block_size;
    unsigned int start_row = block_size * t / size;
    unsigned int start_column = block_size * t % size;
    threads[t] = std::thread(
        f: matmult_thread, std::ref(& a), std::ref(& b), std::ref(& r), count, start_row, start_column);
}
for (std::thread &t : threads)
    t.join();
```


C++ の並列計算パフォーマンス

Parallel Computing Performance of C++

```
void matmult_thread(matrix &a, matrix &b, matrix &r, int count, int start_row, int start_column) {  
    unsigned int size = a.size();  
    int k = start_column;  
    for (int i = start_row; i < size; ++i) {  
        for (; k < size; ++k) {  
            int t = a[i][k];  
            for (int j = 0; j < size; ++j)  
                r[i][j] += t * b[k][j];  
            if (!(--(count)))  
                return;  
        }  
        k = 0;  
    }  
}
```

C++ の並列計算パフォーマンス

Parallel Computing Performance of C++

- ベンチマーク： (parmigiano, gcc with option -O3)

threads	1000	2000	3000	4000	5000
1	0.203070375	3.92559375	13.1114875	29.94645	57.350275
2	0.1072805	1.56977625	5.08685125	11.9222125	22.98695
4	0.059723	1.087904375	3.28801625	7.53337	14.7029
8	0.03474875	0.814612625	2.7524725	6.64562375	13.163225
16	0.021469	0.428517125	1.88304625	4.5489125	10.73210625
32	0.018161625	0.26147475	1.1390575	2.21853125	7.331105

C++ の並列計算パフォーマンス

Parallel Computing Performance of C++

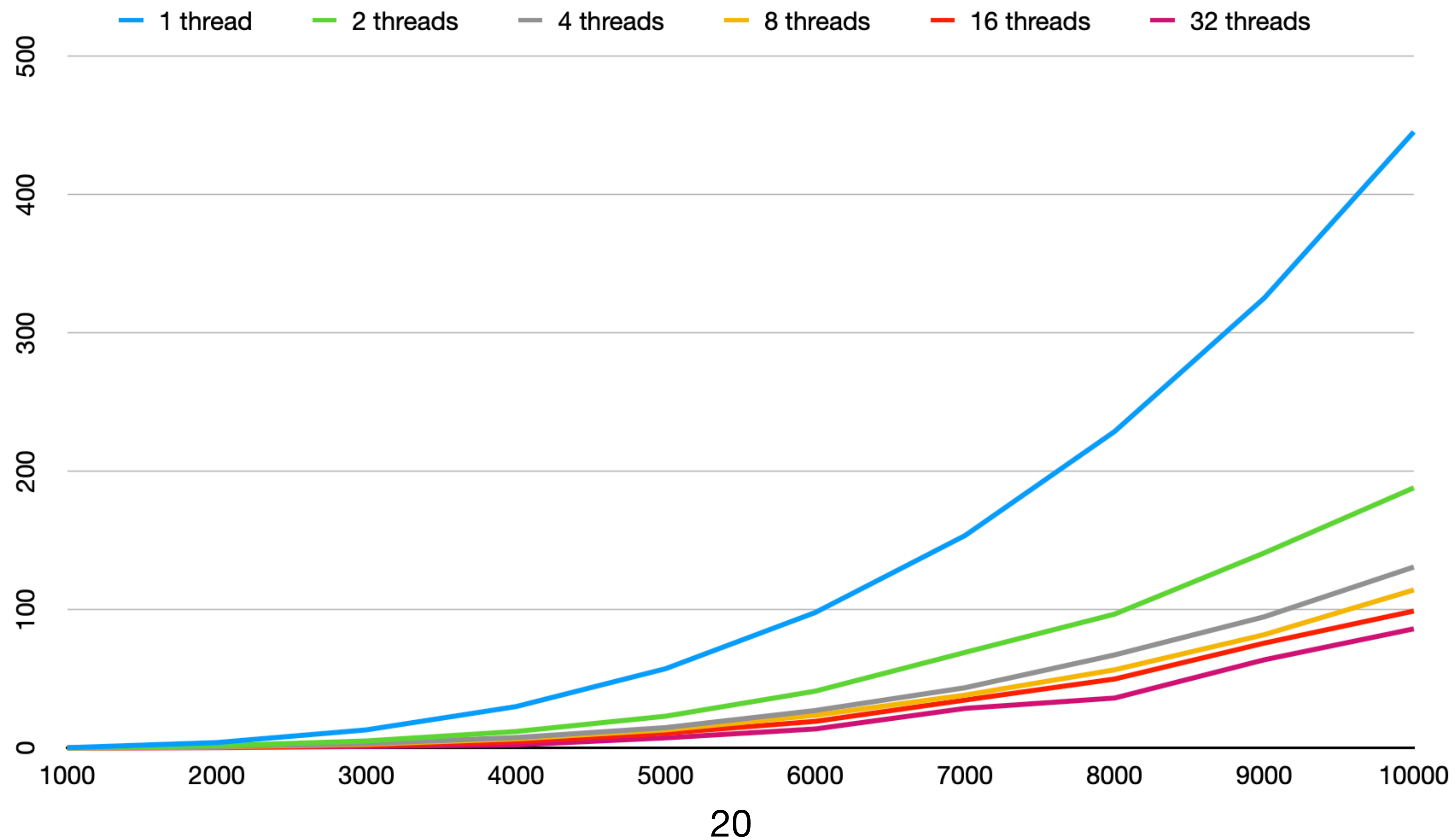
- ベンチマーク： (parmigiano, gcc with option -O3)

threads	6000	7000	8000	9000	10000
1	97.9557	153.5125	228.73325	325.152	445.2935
2	41.091925	69.080225	96.744475	140.951875	188.04425
4	27.077975	43.5442125	67.253475	94.807075	130.790375
8	23.63635	38.193525	56.4852625	81.8788375	114.222625
16	19.253575	34.64125	49.8425	75.7736875	98.9783125
32	13.7497125	28.61185	36.0874875	63.7281375	86.1108125

C++ の並列計算パフォーマンス

Parallel Computing Performance of C++

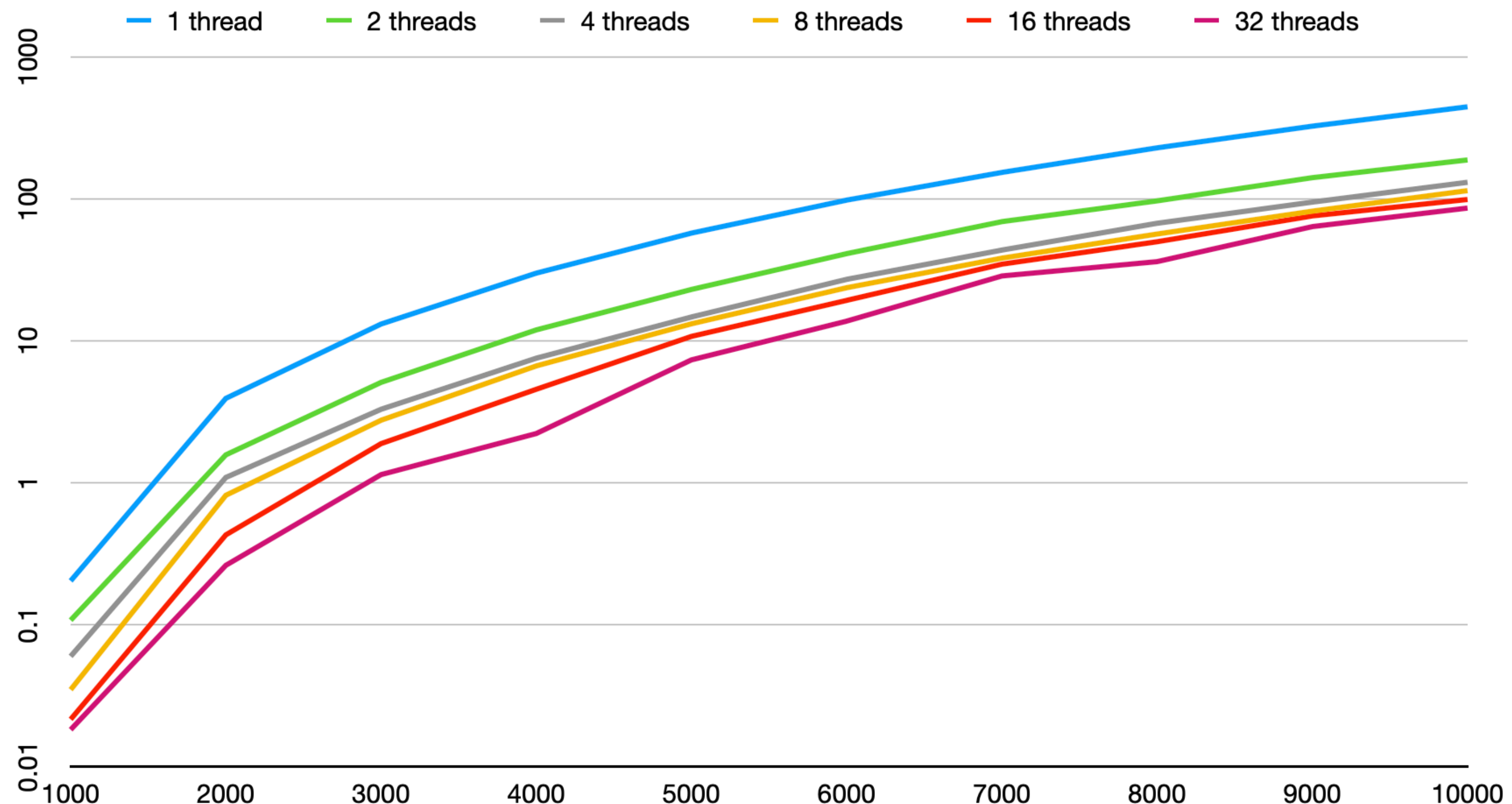
- グラフ:



C++ の並列計算パフォーマンス

Parallel Computing Performance of C++

- グラフ: (Y 軸は対数になる)



1. 先行研究

2. C の並列計算パフォーマンス

3. C++ の並列計算パフォーマンス

4. C と C++ のパフォーマンス比較

5. Nested Pointer 配列について

6. 参考文献

C と C++ のパフォーマンス比較

Performance Comparison of C and C++

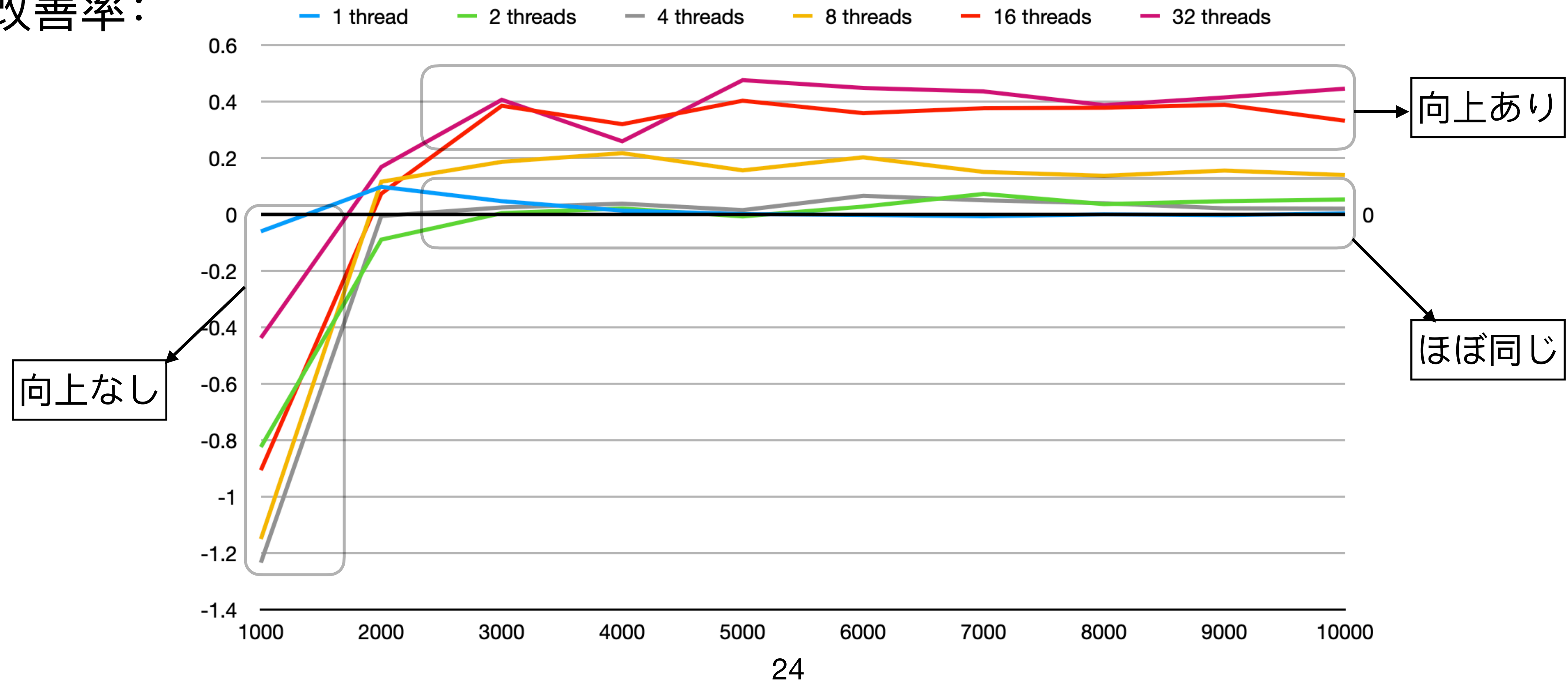
- C プログラムの C++ プログラムに対する改善率:

$$\frac{time_{C-Program} - time_{Cpp-Program}}{time_{Cpp-Program}}$$

C と C++ のパフォーマンス比較

Performance Comparison of C and C++

- 改善率:



C と C++ のパフォーマンス比較

Performance Comparison of C and C++

- 行列のサイズが大きくなり、スレッド数が増える：
 - C プログラムのパフォーマンスはC++ プログラムよりも高くなる
 - パフォーマンスの差はますます大きくなる

1. 先行研究

2. C の並列計算パフォーマンス

3. C++ の並列計算パフォーマンス

4. C と C++ のパフォーマンス比較

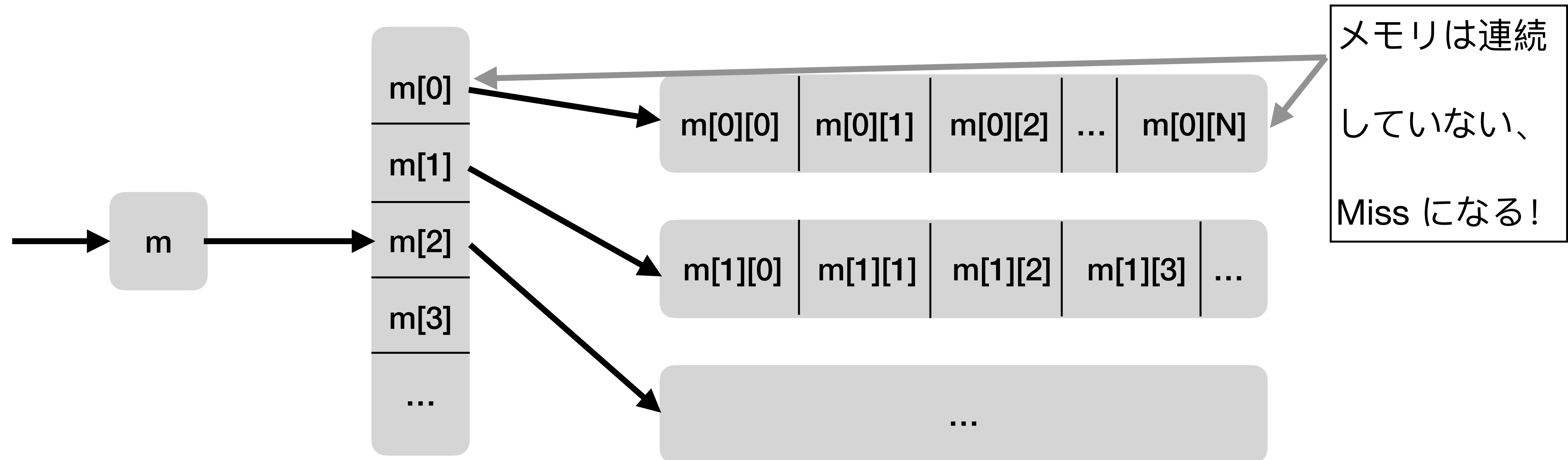
5. Nested Pointer 配列について

6. 参考文献

ネストされたポインタ配列について

About Nested Pointer Arrays

- Nested Pointer 配列はプログラムのパフォーマンスを低下させる



ネストされたポインタ配列について

About Nested Pointer Arrays

- パフォーマンスを向上させるため、データ構造を変える:

```
int **mat;  
// <malloc 省略>  
mat[i][j]; // 行iのj番目の要素
```

メモリが連続するようになる

```
int *mat;  
mat = malloc(sizeof(int) * size * size);  
mat[i * size + j]; // 行iのj番目の要素
```

ネストされたポインタ配列について

About Nested Pointer Arrays

- Nested Pointer Arrays の結果: (parmigiano, gcc with option -O3)

threads	1000	2000	3000	4000	5000
1	0.498162	4.776323	16.076921	37.116737	71.90559
2	0.332681	2.488326	8.203203	18.103214	34.883292
4	0.169404	1.439879	4.371735	9.857323	18.781232
8	0.151378	0.793759	2.486665	5.687532	11.046947
16	0.086189	0.431279	1.288257	3.014409	6.441559
32	0.047484	0.284445	0.9133	1.905143	3.386842

ネストされたポインタ配列について

About Nested Pointer Arrays

- Non-Nested Pointer Arrays の結果: (parmigiano, gcc with option -O3)

threads	1000	2000	3000	4000	5000
1	3.33256	26.314959	87.517885	206.183829	401.777496
2	1.644605	9.219662	29.43524	46.949373	137.247497
4	0.825235	5.794115	16.54925	28.009171	67.803758
8	0.195428	2.549692	7.747583	14.169961	28.855964
16	0.160743	1.463381	4.605367	11.82501	21.641324
32	0.098738	0.843363	2.176077	5.562352	13.140058

ネストされたポインタ配列について

About Nested Pointer Arrays

- Nested Pointer Arrays の結果: (parmigiano, gcc without option -O3)

threads	1000	2000	3000	4000	5000
1	3.639638	29.567218	103.204349	233.92349	467.380447
2	1.918993	14.388551	49.730213	117.507403	231.349138
4	0.876189	7.184843	23.103579	59.706073	113.71306
8	0.468395	4.154684	12.517534	32.383561	61.931366
16	0.29656	2.093115	6.745476	17.15944	33.651
32	0.158688	1.23054	3.697564	9.180414	17.997048

ネストされたポインタ配列について

About Nested Pointer Arrays

- Non-Nested Pointer Arrays の結果: (parmigiano, gcc without option -O3)

threads	1000	2000	3000	4000	5000
1	5.087043	33.448516	131.505275	279.590906	613.235674
2	1.733702	13.440667	42.884861	104.404884	192.946718
4	1.126672	6.405123	22.678031	52.537121	102.351809
8	0.588577	3.743623	12.619298	27.26499	52.591563
16	0.241174	2.122724	6.424687	16.286372	28.423128
32	0.196814	1.144734	3.835803	8.498121	16.953564

ネストされたポインタ配列について

About Nested Pointer Arrays

- Non-Nested Pointer 配列の性能が想像通り表現しない問題は、多分：
 - アルゴリズムと関係があるかもしれない
 - コンパイラの最適化機能と関係があるかもしれない
- 次はこの問題を解決したい

1. 先行研究

2. C の並列計算パフォーマンス

3. C++ の並列計算パフォーマンス

4. C と C++ のパフォーマンス比較

5. Nested Pointer 配列について

6. 参考文献

参考文献

References

- [1] Bryant, R., & O'Hallaron, D. (2019). Computer Systems (pp. 679-683). Harlow, United Kingdom: Pearson Education Limited.
- [2] Sloss, A., Rayfield, J., Symes, D., & Wright, C. (2009). ARM system developer's guide. Amsterdam: Morgan Kaufmann/Elsevier.