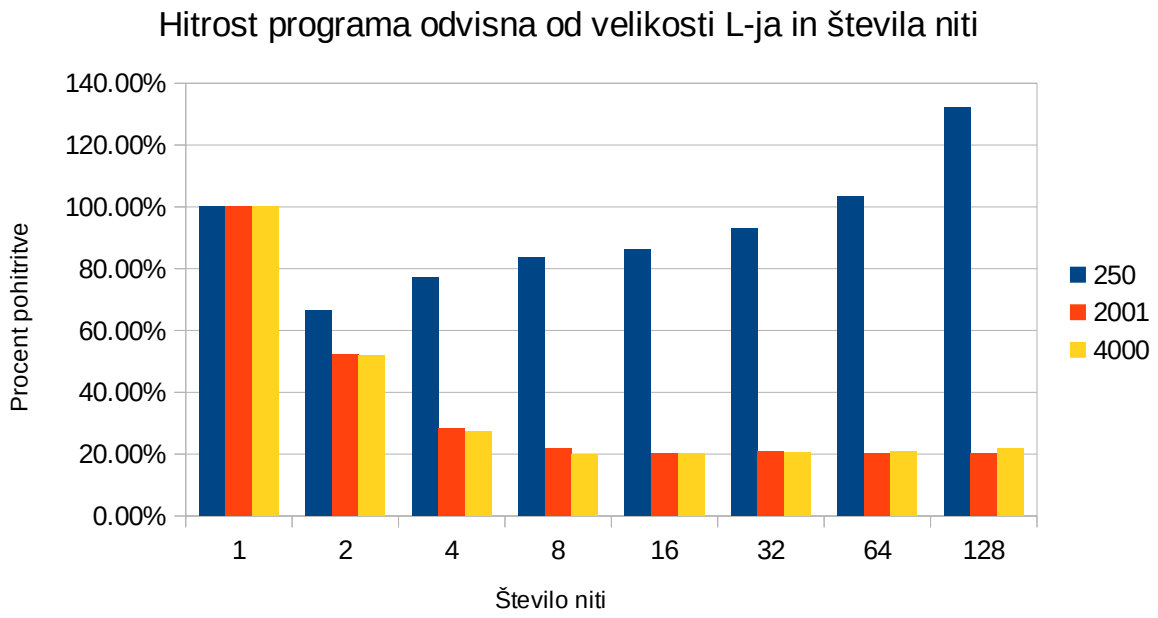
Poročilo N1 – Aleksander Grobelnik



Maksimalna pohititej je okoli 5x, vendar je ta dosežena pri velikih L-jih in ne pri L = 250. Pri L = 250 se pri dveh niti skoraj 2x pohitri, vendar se nato hitro obrne nazaj. Če opazujemo naprej od 8 niti opazimo, da se vrednost pri velikih L-jih ustavi in doseže svoj maksimalni limit pri cca 20%.

Neverjetno je pri majhnih L-ji veliko število niti v bistvu upočasnitev programa, saj se niti dlje časa ukvarjajo z zaklepanjem/odklepanjem mutexa, ki hrani maksimalno število nfes ponovitev. Če bi to ohranili, bi se stvar dosti-krat hitreje obnašala.

Koda programa:

```
#include <algorithm>
#include <array>
#include <climits>
#include <cmath>
#include <iostream>
#include <mutex>
#include <random>
#include <sstream>
#include <thread>
#include <vector>

const int count_threads = 128;
unsigned int nfes = 0;
std::vector<double> mfs;
std::vector<unsigned int> psls;
std::mutex mtx;

void printWrongUsage() {
  std::cout
      << "Usage: main.x -L=<unsign int> -type=[PSL|MF] -seed=<unsigned int> "
        "-nfesLmt=<unsigned int>"
```

```
      << std::endl;
}

int *generateSequence(unsigned int size, unsigned int seed) {
  int *arr = new int[size];
  int values[2] = {-1, 1};
  std::mt19937 mt(seed);
  std::uniform_int_distribution<int> dist(0,
                                          (sizeof(values) / sizeof(int)) - 1);
  for (unsigned int i = 0; i < size; i++) {
    arr[i] = values[dist(mt)];
  }
  return arr;
}

std::string convertBinaryToHexadecimal(int *arr, unsigned int size) {
  int n = ceil((size - 1) / 4) + 1;
  std::string res = "0x";
  for (unsigned int i = n; i > 0; i--) {
    int sum = (*(arr + size - i * 4 + 3) != 1 ? 0 : 1) +
              (*(arr + size - i * 4 + 2) != 1 ? 0 : 1) * 2 +
              (*(arr + size - i * 4 + 1) != 1 ? 0 : 1) * 4 +
              (*(arr + size - i * 4) != 1 ? 0 : 1) * 8;
    res += (sum > 9 ? (char)(sum + 65 - 10) : std::to_string(sum)[0]);
    // std::cout << *(arr + i * 4) << " " << *(arr + i * 4 + 1) << " "
    //      << *(arr + i * 4 + 2) << " " << *(arr + i * 4 + 3) << " " <<
    //      std::endl;
  }
  return res;
}

int C(int *sequence, unsigned int k, unsigned int L) {
  int sum = 0;
  for (unsigned int i = 0; i <= L - k - 1; i++) {
    sum += sequence[i] * sequence[i + k];
  }
  return sum;
}

unsigned int PSL(int *sequence, unsigned int L, unsigned int nfesLmt) {
  int mVal = INT_MIN;
  for (unsigned int k = 1; k < L; k++) {
    if (nfes > nfesLmt) {
      return INT_MIN;
    }
    mVal = std::max(std::abs(C(sequence, k, L)), mVal);
    mtx.lock();
    nfes++;
    mtx.unlock();
  }
  return mVal;
}
```

```
double MF(int *sequence, unsigned int L, unsigned int nfesLmt) {
  int energy = 0;
  for (unsigned int k = 1; k < L; k++) {
    if (nfes > nfesLmt) {
      return 0;
    }
    int ck = C(sequence, k, L);
    energy += ck * ck;
    mtx.lock();
    nfes++;
    mtx.unlock();
  }
  return (L * L) / (2.0 * energy);
}

int split(int *arr, std::string type, int a, int b) {
  double pivot = arr[a];
  int li = a;
  int ri = b;
  while (li < ri) {
    while (arr[li] <= pivot && li < b) {
      li++;
    }
    while (arr[ri] >= pivot && ri > a) {
      ri--;
    }
    if (li < ri) {
      std::swap(arr[li], arr[ri]);
    }
  }
  std::swap(arr[a], arr[ri]);
  return ri;
}

void quicksort(int *arr, std::string type, int a, int b) {
  if (a < b) {
    int splitter = split(arr, type, a, b);
    quicksort(arr, type, a, splitter - 1);
    quicksort(arr, type, splitter + 1, b);
  }
}

void computeSequence(int *sequence, std::string type, unsigned int size,
                     unsigned int start, unsigned int end,
                     unsigned int nfesLmt) {
  for (unsigned int i = start; i <= end; i++) {
    int *seq = new int[size];
    for (unsigned int i = 0; i < size; i++) {
      seq[i] = sequence[i];
    }
    seq[i] *= -1;
```

```
    if (type == "PSL") {
      int psl = PSL(seq, size, nfesLmt);
      psls[i] = psl;
    } else if (type == "MF") {
      double mf = MF(seq, size, nfesLmt);
      mfs[i] = mf;
    }
  }

  if (end == size - 1) {
    if (type == "PSL") {
      int psl = PSL(sequence, size, nfesLmt);
      psls[size] = psl;
    } else if (type == "MF") {
      double mf = MF(sequence, size, nfesLmt);
      mfs[size] = mf;
    }
  }
}

int main(int argc, char **argv) {
  if (argc == 1) {
    std::cout << "Wrong number of given arguments" << std::endl;
    printWrongUsage();
    return 1;
  }
  unsigned int seed = 0;
  unsigned int L = 0;
  std::string type = "";
  unsigned int nfesLmt = 0;
  for (int i = 1; i < argc; i++) {
    std::string arg(argv[i]);
    int pos = arg.find("=");
    if (pos == -1 || arg == "-h" || arg == "-?") {
      printWrongUsage();
      return 1;
    }
    if (arg.substr(0, pos) == "-L" || arg.substr(0, pos) == "L") {
      L = stoi(arg.substr(pos + 1));
    } else if (arg.substr(0, pos) == "-type" || arg.substr(0, pos) == "type") {
      type = arg.substr(pos + 1);
    } else if (arg.substr(0, pos) == "-seed" || arg.substr(0, pos) == "seed") {
      seed = stoi(arg.substr(pos + 1));
    } else if (arg.substr(0, pos) == "-nfesLmt" ||
               arg.substr(0, pos) == "nfesLmt") {
      nfesLmt = stoi(arg.substr(pos + 1));
    }
  }
  if (seed == 0 || L == 0 || type == "" || nfesLmt == 0) {
    std::cout << (seed != 0 ? "FOUND: seed" : "NOT FOUND: seed") << std::endl
              << (L != 0 ? "FOUND: L" : "NOT FOUND: L") << std::endl
              << ((type == "MF" || type == "PSL") ? "FOUND: type"
```

```
                                        : "NOT FOUND: type")
          << std::endl
          << (nfesLmt != 0 ? "FOUND: nfesLmt" : "NOT FOUND: nfesLmt")
          << std::endl;
  printWrongUsage();
  return -1;
}
std::chrono::time_point<std::chrono::system_clock> start, end;

// int arr[] = {1, 1, 1, -1, -1, -1, 1, -1, -1, 1, -1, -1};
int *sequence = generateSequence(L, seed);
psls.resize(L + 1);
mfs.resize(L + 1);
std::array<std::thread, count_threads> threads;
int thr_i = 0;
start = std::chrono::system_clock::now();
for (auto &thr : threads) {
  unsigned int start = thr_i * (L / count_threads);
  unsigned int end = thr_i == (count_threads - 1)
                     ? L - 1
                     : (thr_i + 1) * (L / count_threads);
  thr = std::thread([sequence, type, L, start, end, nfesLmt] {
    computeSequence(sequence, type, L, start, end, nfesLmt);
  });
  thr_i++;
}
for (auto &thr : threads) {
  thr.join();
}
end = std::chrono::system_clock::now();
std::chrono::duration<double> elapsed_seconds;
elapsed_seconds = end - start;
unsigned int index;
if (type == "MF") {
  index = std::distance(std::begin(mfs),
                        std::max_element(std::begin(mfs), std::end(mfs)));
} else {
  index = std::distance(std::begin(psls),
                        std::min_element(std::begin(psls), std::end(psls)));
}
sequence[index] *= -1;
std::cout << "L: " << L << std::endl
          << "nfesLmt: " << nfesLmt << std::endl
          << "seed: " << seed << std::endl
          << "nfes: " << nfes << std::endl
          << // Število ovrednotenj
    "runtime: " << elapsed_seconds.count() * 1000 * 1000 << "qs" << std::endl
          << //Čas izvajanja algoritma
    "speed: " << nfes / elapsed_seconds.count() * 1000 * 1000 << std::endl
          << // Število ovrednotenj na sekundo
    "sequence: " << convertBinaryToHexadecimal(sequence, L) << std::endl
          << "MF: " << mfs[index] << std::endl
```

```
            << "PSL: " << psls[index] << std::endl;
  /*std::cout << "Operations per second: "
       << size * (method != 2 ? 1 : size) / elapsed_seconds.count() <<
  std::endl; std::cout << "Number of operations: " << size * (method != 2 ? 1 :
  size) << std::endl;*/
  // delete[] sequence;
  return 0;
}
```

V spodni preglednici tudi dodajam neobdelane podatke iz katerih sem naredil zgornja dva grafa.

count of threads | L	250	2001	4000 in µs
1	15932	1808540	3593480
2	10618	944317	1862100
4	12304	511725	982662
8	13341	391873	709298
16	13736	365541	720870
32	14824	373789	735301
64	16466	368336	750876
128	21075	367354	785694

count of threads | L	250	2001	4000 in µs
	15932	1808540	3593480
1	100.00%	100.00%	100.00%
2	66.65%	52.21%	51.82%
4	77.23%	28.29%	27.35%
8	83.74%	21.67%	19.74%
16	86.22%	20.21%	20.06%
32	93.05%	20.67%	20.46%
64	103.35%	20.37%	20.90%
128	132.28%	20.31%	21.86%