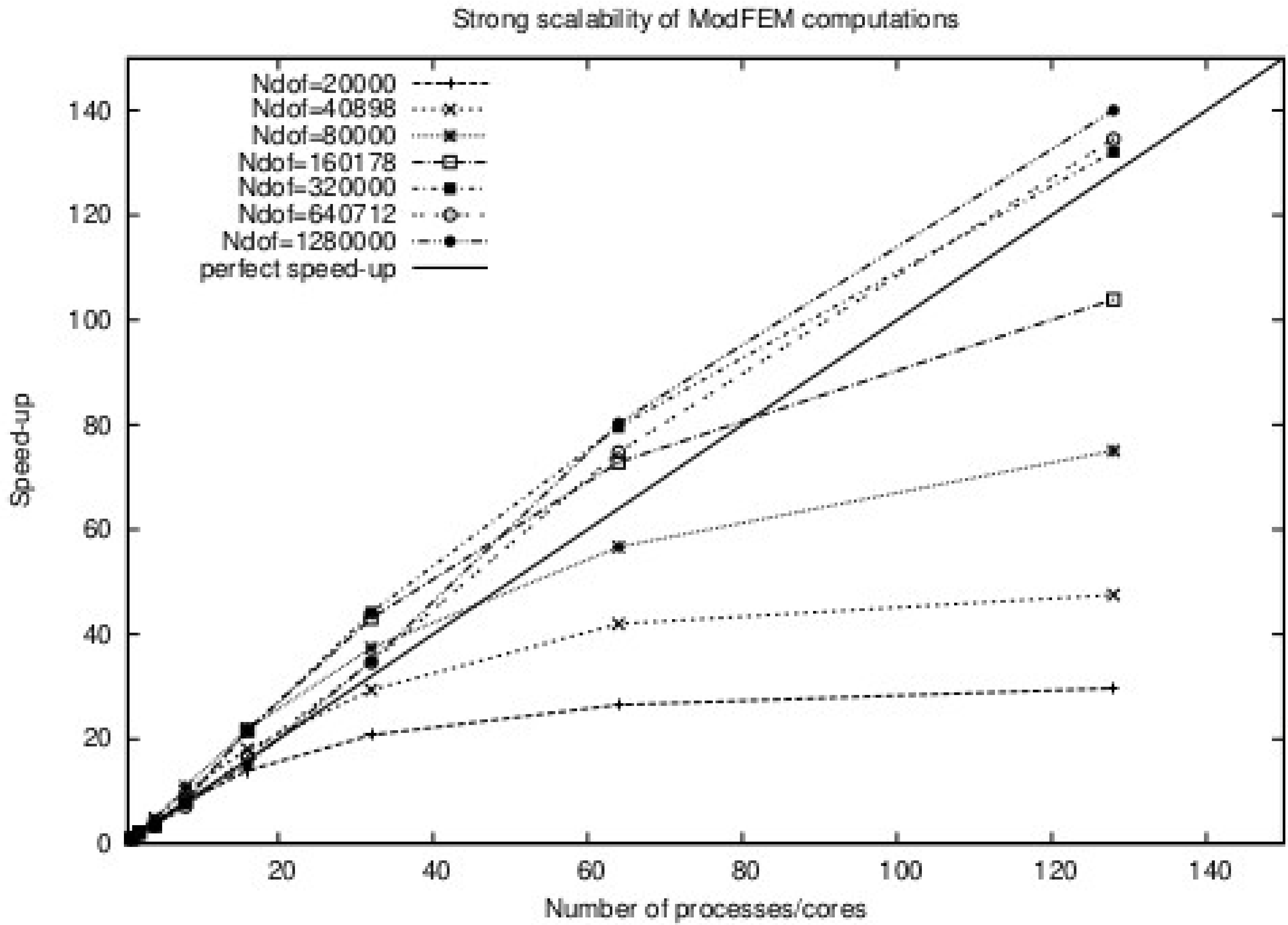
Analysis and modeling of

Computational Performance

Scalability

- Standard parallel performance measures
  - speed-up
  - efficiency

  characterise the so called strong scalability of programs
- Good strong scalability, being closed to the perfect, linear speed-up, is difficult to obtain, e.g. due to:
  - sequential (not possible to be parallelized) parts of the program/algorithm (as in Amdahl law)
  - communication
- It is possible to obtain good parallel performance also for programs with sequential parts and communication
  - for many algorithms/programs the speed-up curves become closer to the perfect speed-up with the increasing problem size
    - the most popular way for expressing problem size is the number of (dominant) operations – the operations for which the execution time is proportional to the number of operations



Strong scalability of ModFEM computations
Ndof=20000
Ndof=40898
Ndof=80000
Ndof=160178
Ndof=320000
Ndof=640712
Ndof=1280000
perfect speed-up
Speed-up
Number of processes/cores
0
20
40
60
80
100
120
140

Scalability

- Strong scalability measures the performance for the increasing number of resources used in computations (processors, computational nodes)
- The standard notions of scalability include also the condition of increasing workload
- The increasing workload in case of programs may be the number of dominant operations
  - sometimes the execution time for single processor/core is taken as the workload
  - in the simplified analysis the time per single dominant operation is constant – so the number of dominant operations is proportional to the execution time for single processor/core
- Weak scalability denotes the scalability for the case where the total workload is not constant (as for the strong scalability), but the workload per single processor/core (thread/process) is constant
  - constant workload per thread/process means the total workload proportional to the number of threads/processes

Weak scalability

- For weak scalability study, one can define the execution time as the function of the number of threads/processes and the workload
  - $T_{||} = T_{||}(p, W) = (\text{for weak scalability})\ T_{||}(p, pW_0)$
- Then it is possible to define the scaled speed-up, $\boldsymbol{S^S(p)}$:
  - $\boldsymbol{S^S(p) = T_{||}(1, pW_0) / T_{||}(p, pW_0) = p * T_{||}(1, W_0) / T_{||}(p, pW_0)}$
- A program/algorithm has linear weak scalability when its scaled speed-up is linear
- Linear weak scalability is equivalent to:
  - **the same execution time for p-times larger problems executed on p cores/processors**
  - the parallel overhead constant per single processor/core for p-times larger problems executed on p cores/processors

Scalability of computations

- Scalability is the key property for obtaining high performance of computations
- Weak scalability (almost linear) can be attributed to many algorithms, while linear strong scalability is extremely rare
- The general relation:
  - *performance = number_of_operations / execution_time*

  can be transformed to the expression:
  - *performance = speed-up / execution_time_per_single_operation*
    - good speed-up denotes effectively parallelized programs
    - short *execution_time_per_single_operation* characterises well optimized single thread computations
    - hence the total performance has two ingredients:
      - scalability
      - single thread performance

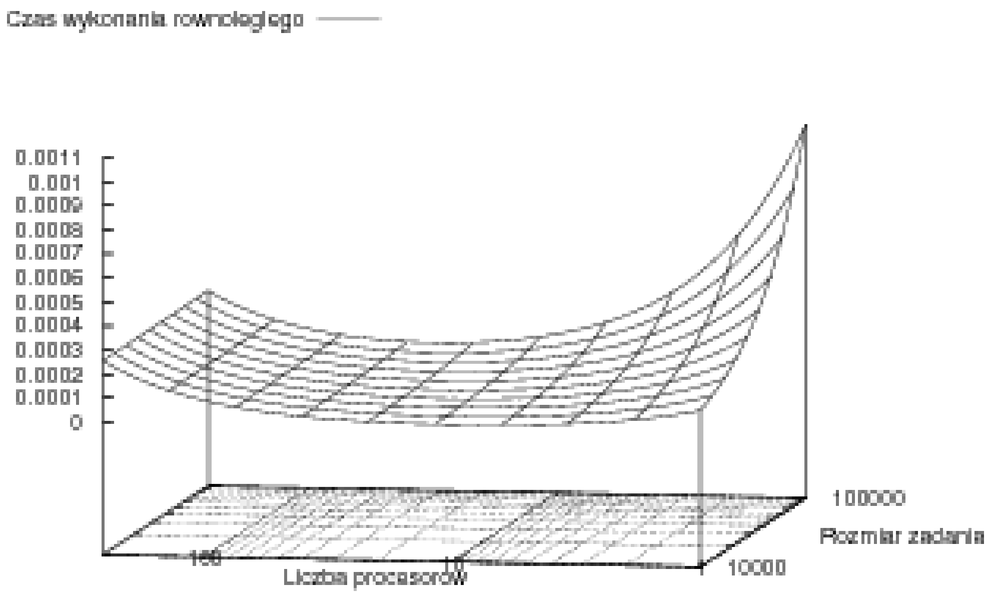
Execution time modelling

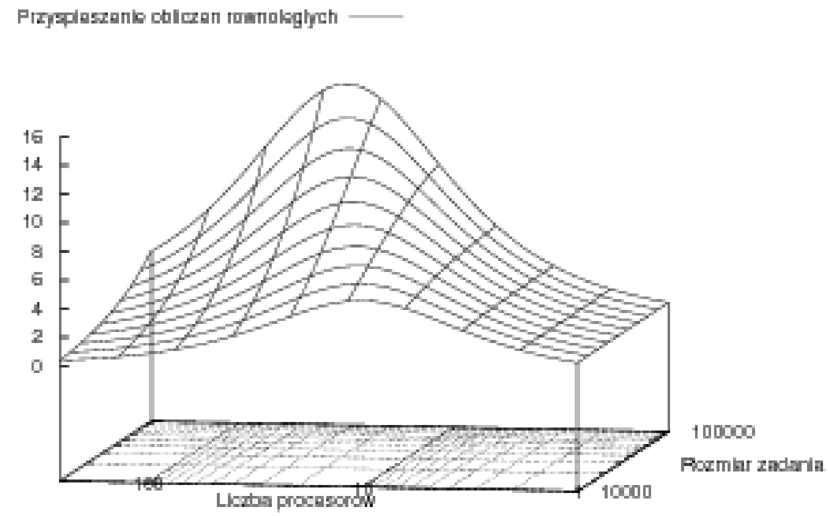
- When modelling execution time several simplifying assumptions can be adopted::
  - The single thread execution time and the computation time for a single thread in parallel programs are proportional to the number of dominant operations, which for the considered algorithms are the arithmetic operations
    - the notion of arithmetic intensity makes also the execution time of programs with memory limited performance proportional to the number of arithmetic operations
  - The time for performing a single dominant operation is constant and denoted by t_c
    - t_c is some amortized time per operation that includes memory accesses, sequential execution system overhead (e.g. memory allocation), etc.
  - Apart from t_c there are only two other hardware parameters, that characterise the communication time: t_s and t_w
  - The parallelized computations are perfectly balanced

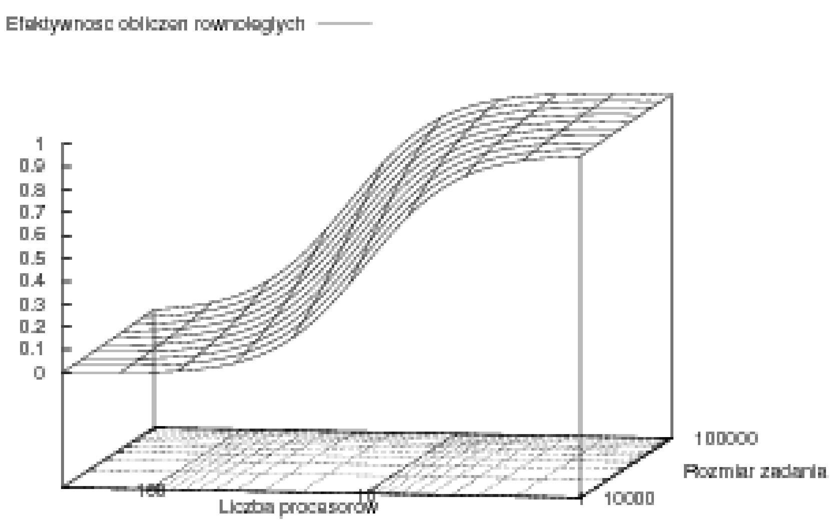
Example

- Calculation of the norm of vector with size N
- N additions and multiplications – decomposition to obtain local sums – perfect speed-up possible
- Global sum reduction – communication time dominates arithmetic operations time
- Naive algorithm – all threads/processes send their local sums to the master thread/process
- Execution time modelling: $T_{\|}(p) = 2*N*t_c/p + p*(t_s+8*t_w)$
- Workload (number of operations): $W = 2*N$
- The same workload per thread/process for week scalability study:
  - $W(p) = W_1 * p = 2 * N_1 * p$
  - $T_{\|}(p) = 2*N*t_c/p + p*(t_s+8*t_w) = 2 * N_1 * t_c + p * (t_s+8*t_w)$
- Simple analysis to obtain: speed-up, efficiency, scaled speed-up, scaled efficiency, isoefficiency function, memory size limited speed-up, etc.

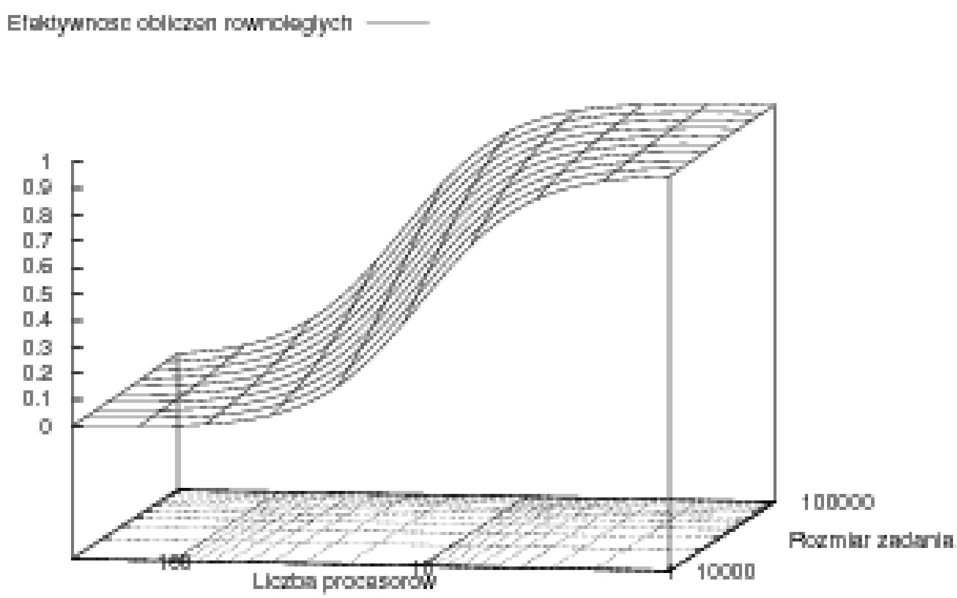
Example

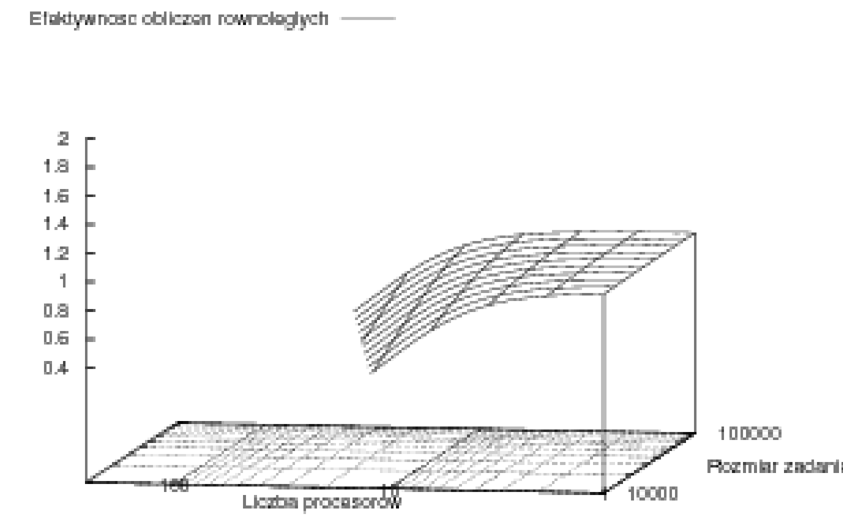

Czas wykonania rownoleglego
0.0011
0.001
0.0009
0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0
100000
Rozmiar zadania
10000
Liczba procesorow

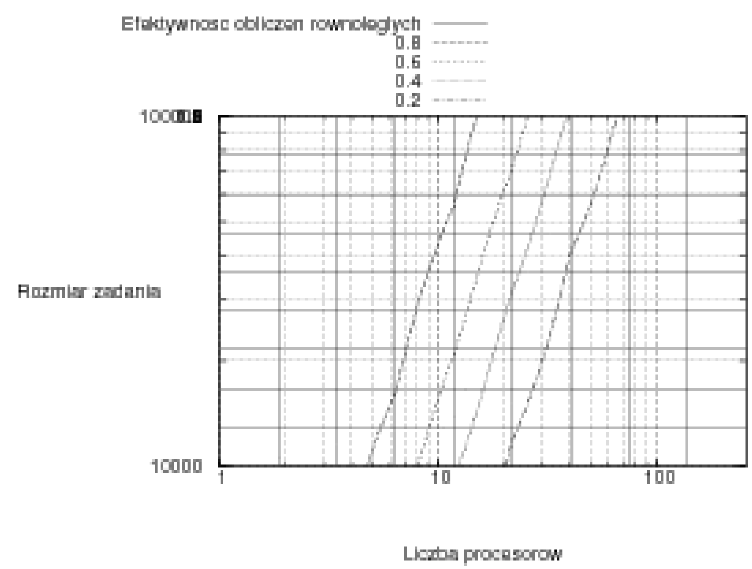

Przyspieszenie obliczen rownoleglych
16
14
12
10
8
6
4
2
0
100000
Rozmiar zadania
10000
Liczba procesorow


Efektywnosc obliczen rownoleglych
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
100000
Rozmiar zadania
10000
Liczba procesorow

Example


Efektywnosc obliczen rownoleglych
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
100000
Rozmiar zadania
10000
Liczba procesorow


Efektywnosc obliczen rownoleglych
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
100000
Rozmiar zadania
10000
10
Liczba procesorow


Efektywnosc obliczen rownoleglych
0.8
0.6
0.4
0.2
Rozmiar zadania
10000
1
10
100
Liczba procesorow

Optimization of parallel programs

- To minimize execution time for parallel programs, for distributed memory computer architectures, the following steps should be undertaken:
  - load balancing
  - minimization of the total size of messages
  - minimization of the number of messages (by increasing the granularity of computations)
  - avoiding network contention
  - reducing the time for additional operations related to parallel computations (e.g. redundant computations – but redundant computations can decrease the communication volume)
  - reducing system overhead (e.g. for synchronization)
  - overlapping computations with communication
  - optimizing single thread execution time
    - including optimizing memory accesses