# Последовательно-параллельное преобразование для расчета Надёжности Сетей с Ограничением на Диаметр

Мигов Д.А.
Нестеров С.Н.

ИВМиМГ

20 сентября 2017 г.

# Введение

Пусть $G = (V, E)$ – граф, в котором задано множество терминальных вершин $K \subseteq V$.

$K$-терминальная надёжность графа $G$ с ограничением на диаметр $d$, $R_K(G, d)$, определяется как вероятность того, что для каждой пары вершин $u, v \in K$ существует надёжный путь длины не более $d$ рёбер.

# Сложность

Таблица: Сложность вычислений при различных значений диаметра $d$ и количества терминалов $k = |K|$

| | | $\longleftarrow$ | $k$ | $\longrightarrow$ |
|---|---|---|---|---|
| | | 2 | $3 \dots n-1$ | $n$ |
| | 2 | O(n) - 2004 | O(n) - 2013 | $\mathcal{NP}$-hard - 2014 |
| $\uparrow$ | 3 | | | |
| d | $\vdots$ | $\mathcal{NP}$-hard - 2004 | | $\mathcal{NP}$-hard - 2014 |
| $\downarrow$ | $n-2$ | | | |
| | $\geq n-1$ | $\mathcal{NP}$-hard - 1977 | | $\mathcal{NP}$-hard - 1983 |

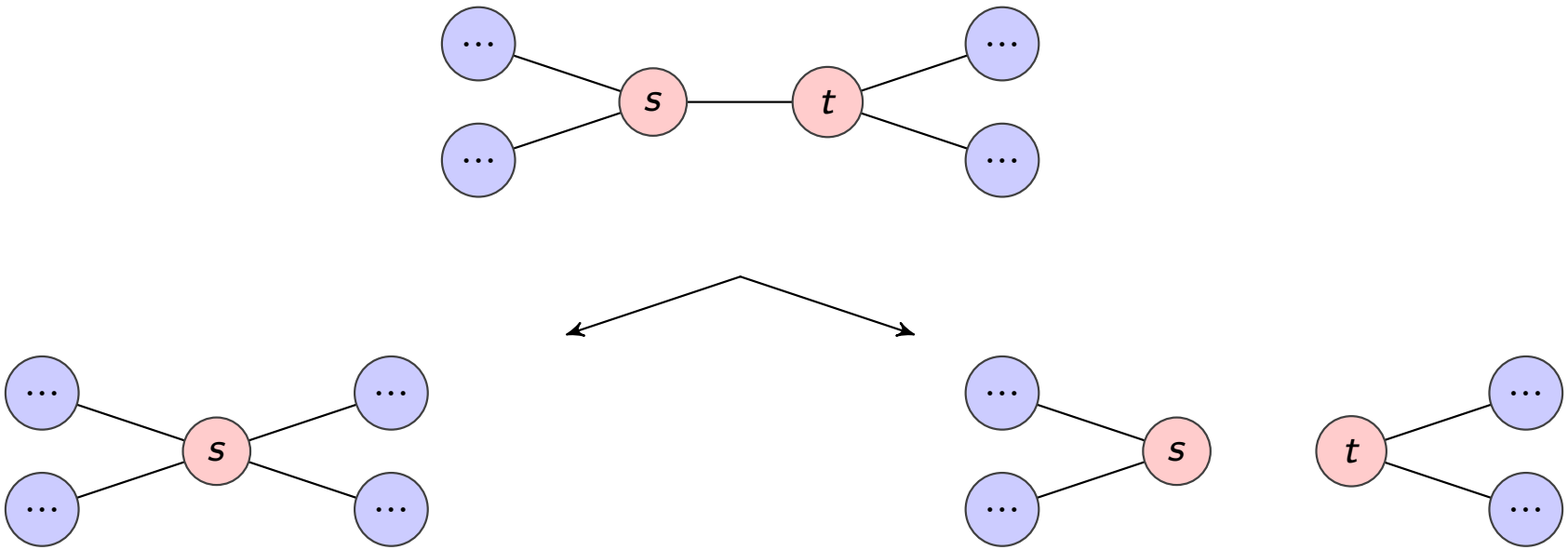
# Формула факторизации

Теорема факторизации Шеннона-Мура (1956)

$$R(G) = r_e \times R(G \backslash e) + (1 - r_e) \times R(G/e),$$

где

- $e = \{s, t\} \in E$,
- $G/e$ — сеть, образованная из $G$ удалением ребра $e$ из $E$,
- $G \backslash e$ — сеть, в которой ребро $e$ стягивается,
- $r_e = r_{s,t} \in [0, 1]$ – вероятность того, что канала связи $e$ не откажет.



# Формула факторизации

В случае $K$-терминальной надёжности с ограничением на диаметр **формула факторизации** для графа $G = (V, E)$ принимает следующий вид:

$$R_K(G, d) = r_e \times R_K(G \backslash e, d) + (1 - r_e) \times R_K(G/e, d),$$

где

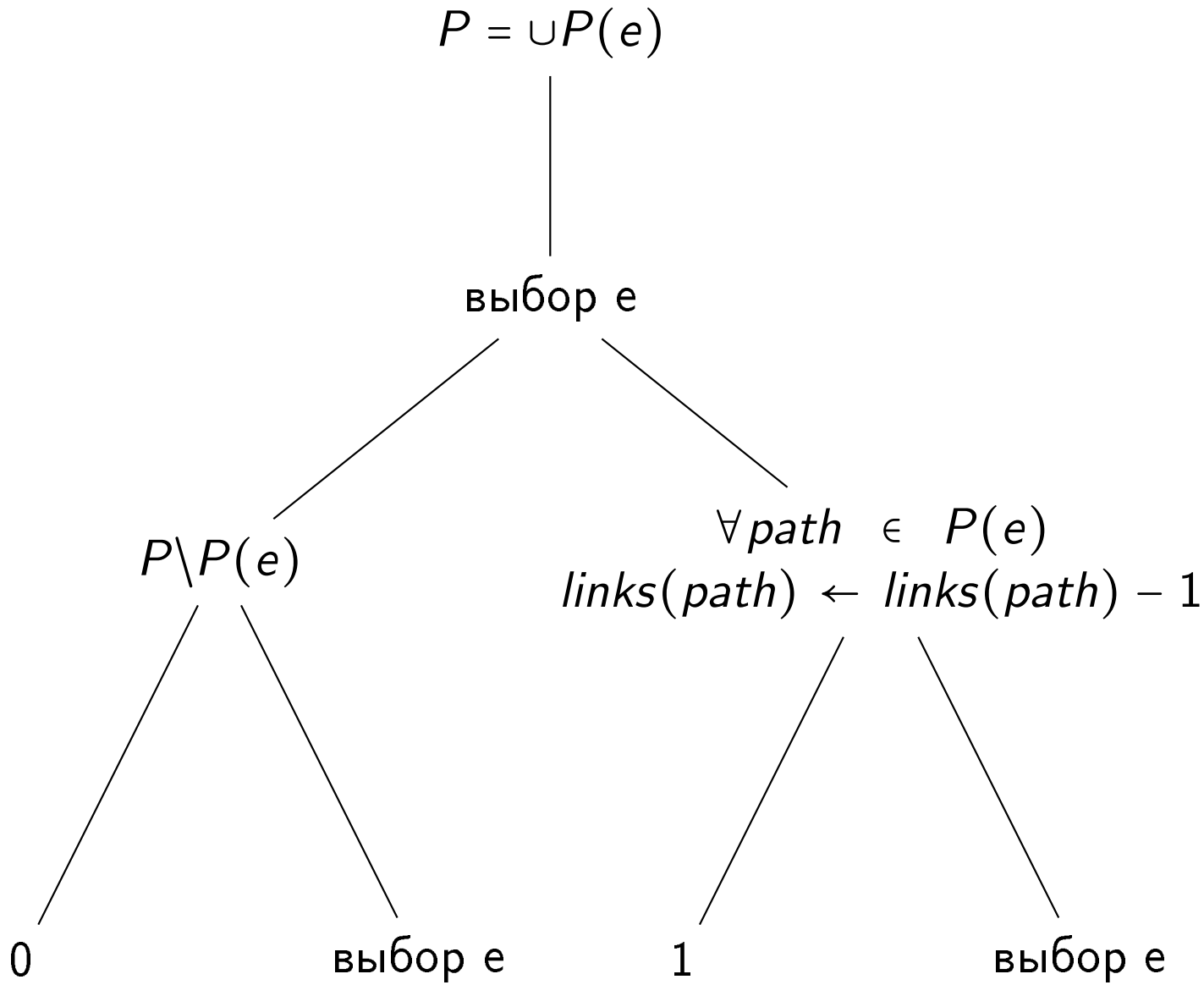
- $G/e$ – сеть, образованная из $G$ удалением канала связи $e$ из $E$,
- $G \backslash e$ – сеть, в которой вероятность канала $e$ устанавливается равной 1.

# алгоритм Cancela и Petingi

Оптимальнее будет использование алгоритма, опубликованного Cancela и Petingi.
Для этого нам понадобятся некоторые дополнительные структуры:

- $P_{s,t}(d)$ – множество всех путей между $s$ и $t$, длина которых не превосходит $d$,
- $P(d)$ – объединение всех множеств $P_{s,t}(d)$ для любых $s, t \in K$,
- $P(e)$ – список всех путей $p \in P(d)$, содержащих в себе ребро $e$,
- $links_p$ – число рёбер в пути $p \in P$, не являющихся абсолютно надёжными.

# Схема алгоритма Cancela и Petingi



# Обновление оценок надёжности

Предположим, что заданная сеть $G$ в процессе факторизации приведена в $L$ подсетей $G_1, \ldots, G_L$. Пусть $P_l$ при $l \in \{1, \ldots, L\}$ - вероятность получения $G_l$. Тогда $\sum_{l=1}^{L} P_l = 1$ и

$$R(G) = \sum_{l=1}^{L} P_l R(G_l) = 1 - \sum_{l=1}^{L} P_l (1 - R(G_l))$$

Таким образом:

$$\sum_{l=1}^{k} P_l R(G_l) \leq R(G) \leq 1 - \sum_{l=1}^{k} P_l (1 - R(G_l))$$

Введём обозначения $RL_l$ и $RU_l$ для $G_1, \ldots, G_L$ и установим значения $RL_0$ и $RU_0$ нулём и единицей соответственно.

$$RL_l = RL_{l-1} + P_l R(G_l),$$
$$RU_l = RU_{l-1} - P_l (1 - R(G_l)).$$

# Эксперименты. 47 вершин и 69 рёбер.

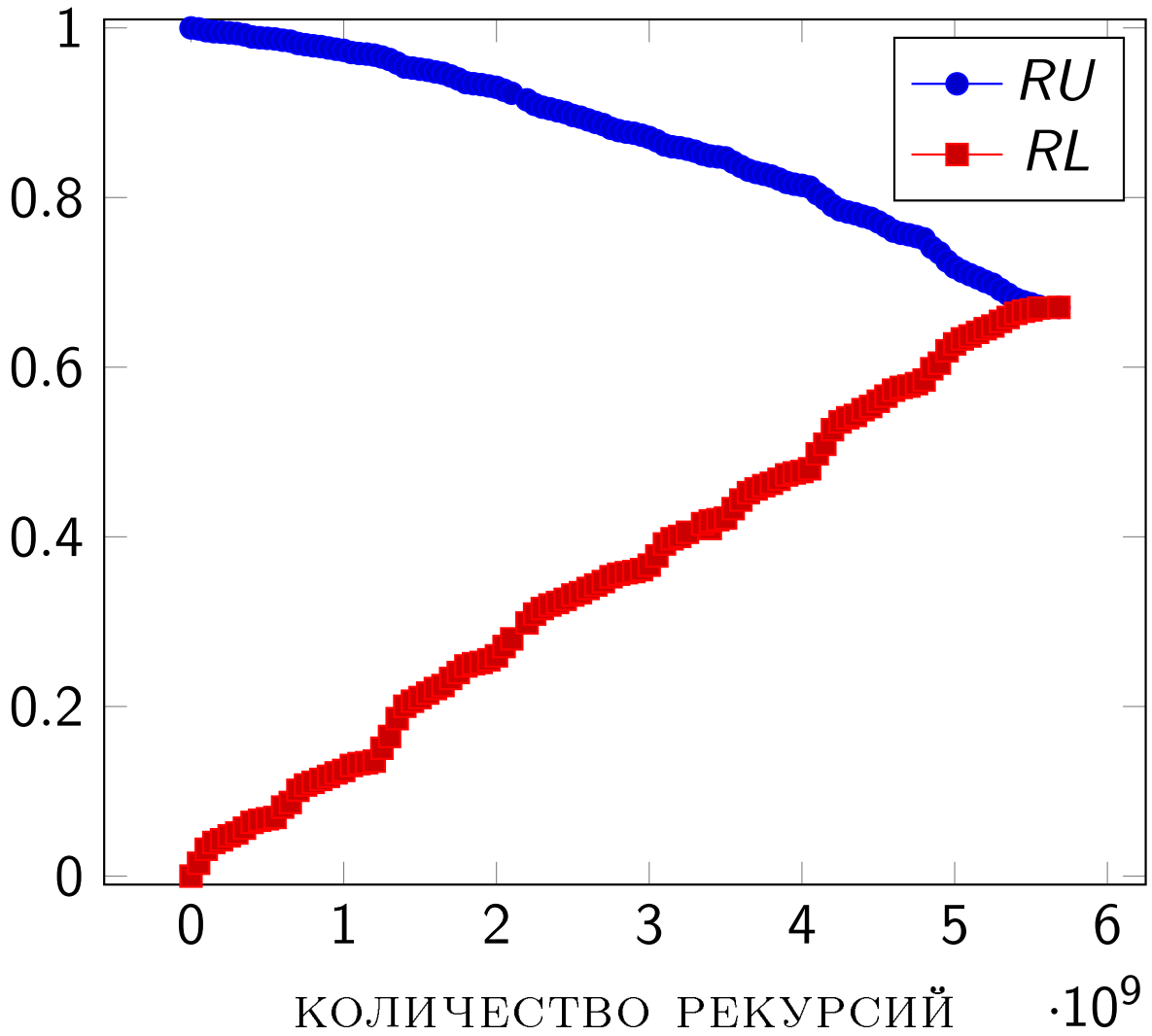
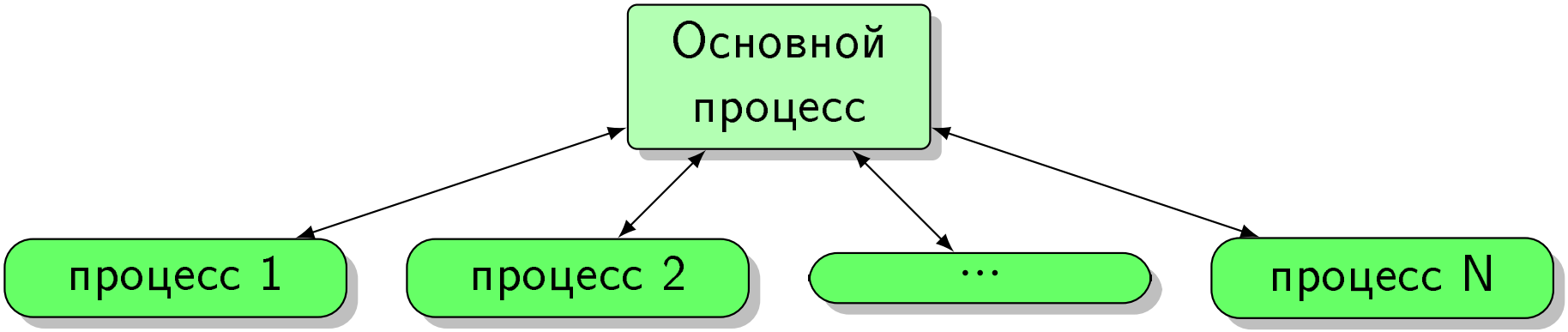


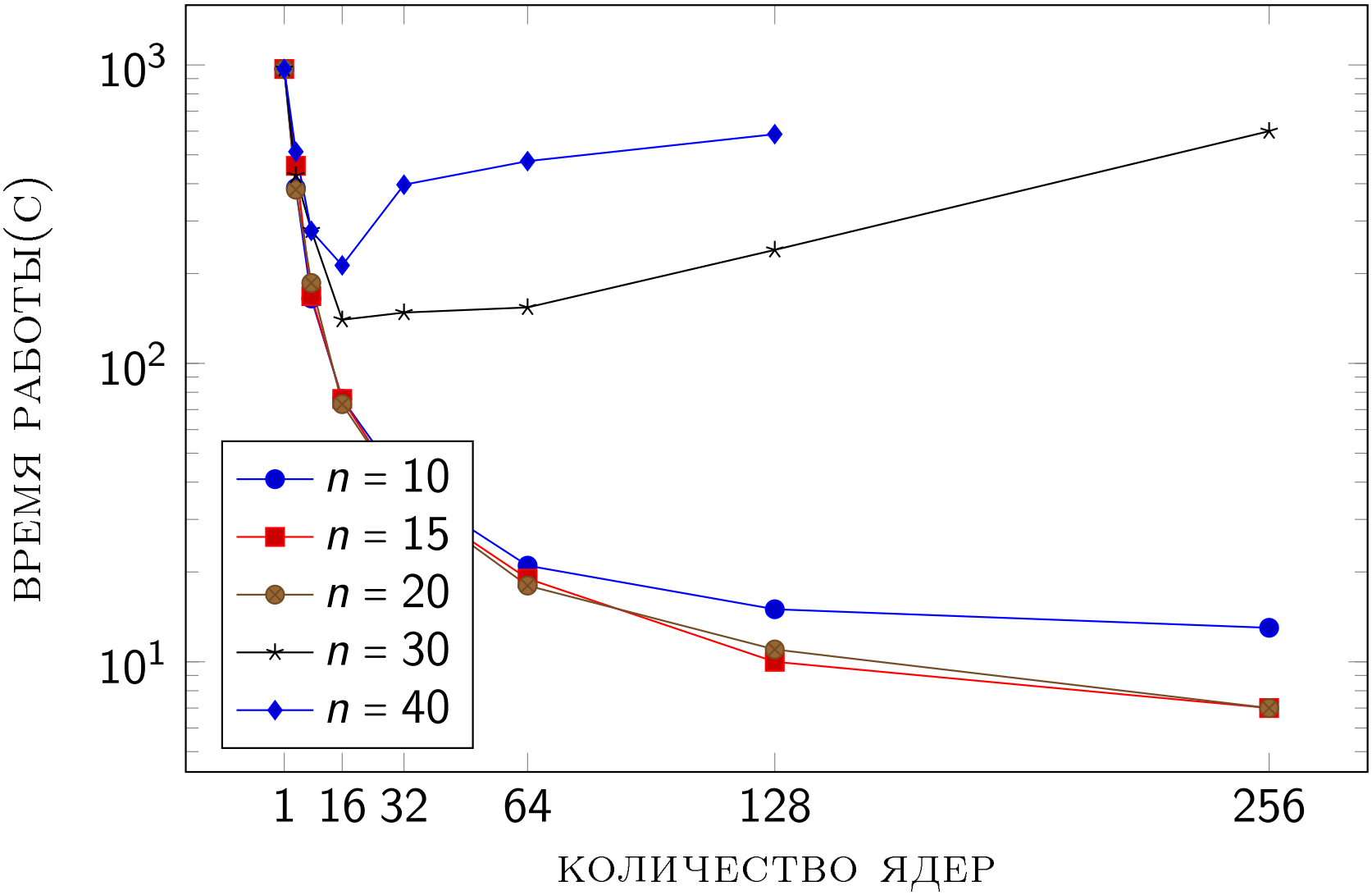

Рис.: $d = 20,\ P = 0.7,\ T \approx 59,5$ часов

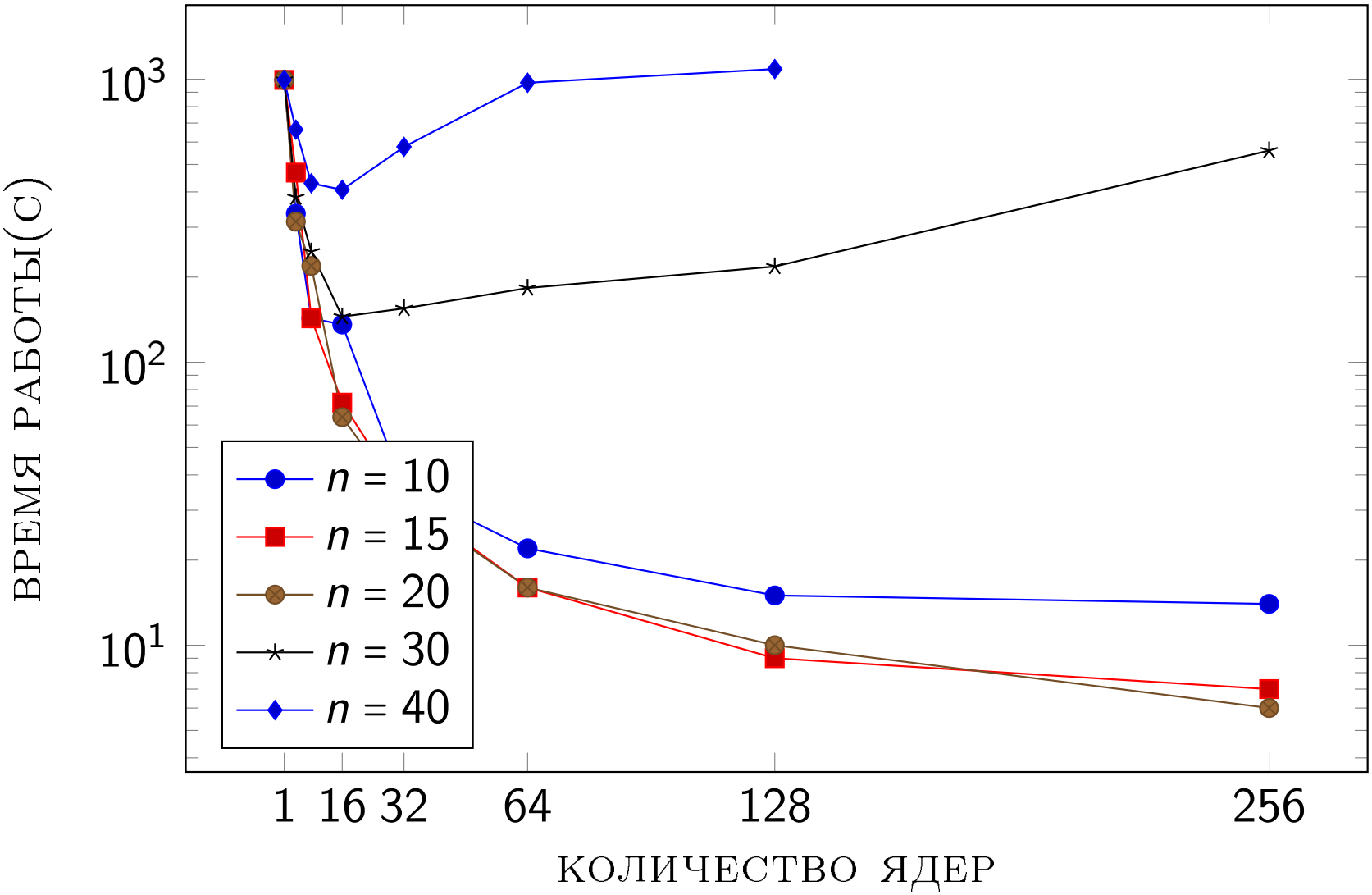
# Фабрика процессов (ведущий-ведомые процессы)




- Основной процесс последовательно выполняет предварительную часть программы и запускает ряд подчинённых процессов для выполнения параллельной работы,
- Когда работник закончил работу, он информирует главный процесс, который может отправить ему новую задачу,
- При необходимости работник обращается к основному процессу, который выделяет один из простаивающих процессов для помощи.

# simple Factoring




ВРЕМЯ РАБОТЫ(С)
$10^3$
$10^2$
$10^1$
1 16 32 64 128 256
КОЛИЧЕСТВО ЯДЕР
$n = 10$
$n = 15$
$n = 20$
$n = 30$
$n = 40$

# C&P, уточняющий оценки

```
1 Function FACTO(Input, P_l)
2 |   if nowT − startT > limitT or RL > fixVal or RU < fixVal
  |    then
3 |   |   return
4 |   end
5 |   e ← ребро : 0 < r_e < 1
6 |   contractEdge(Input, e, P_l) // случай, отвечающий
  |    стягиванию ребра
7 |   deleteEdge(Input, e, P_l) // случай, отвечающий
  |    удалению ребра
8 end
```

# параллельный алгоритм

```
1  Function MPIFACTO(Input, P_l)
2      e ← ребро
3      send(helpMessage) to master process
4      receive(helper) from master process
5      if helper = 0 then
6          contractEdge(Input, e, P_l) // случай, отвечающий
           стягиванию ребра
7      end
8      else
9          send(Input, e, P_l) to helper //отправляем данные
           помощнику
10     end
11     deleteEdge(Input, e, P_l) // случай, отвечающий
       удалению ребра
12 end
```

# параллельный алгоритм (NEW)

```
1  Function MPIFACTO(Input, P_l)
2      e ← ребро
3      send(helpMessage) to master process
4      receive(helper) from master process
5      if helper = −1 then
6          return
7      end
8      if helper = 0 then
9          contractEdge(Input, e, P_l) // случай, отвечающий
           стягиванию ребра
10     end
11     else
12         send(Input, e, P_l) to helper //отправляем данные
           помощнику
13     end
14     deleteEdge(Input, e, P_l) // случай, отвечающий
       удалению ребра
15 end
```

# параллельный алгоритм

```
1  Function MASTERPROCESS()
2      calculatePst(Input) // считываем все данные и
        формируем нужные списки
3      send(Input) to process 1 // запускаем первый процесс
4      while есть занятые процессы do
5          receive(message) from sender //получаем
            сообщение
6          if message = I_AM_FREE then
7              sender ← idle status //
8          end
9          else if message = I_NEED_HELP then
10             helper ← 0 or idle process
11             send(helper) to sender
12         end
13     end
14     send(STOP) to slaves
15     recursionCount ← sumFromSlaves(recursionCount)
```

# параллельный алгоритм

```
1  Function SLAVEPROCESS()
2      recursionCount ← 0
3      R_K^D(G) ← 0
4      do
5          receive(message) from sender //получаем
            сообщение
6          if message = MPI_FACTO then
7              receive(Input) from sender
8              MPIFACTO(Input, 1)
9          end
10         else if message = CONTRACT_EDGE then
11             receive(Input, e) from sender
12             contractEdge(Input, e, P_l)
13         end
14     while message ≠ STOP;
15     send(recursionCount, R_K^D(G)) to master process
16 end
```

# C&P, уточняющий оценки

```
1  Function contractEdge(Input, e, P_l)
2      P_l ← P_l * r_e
3      foreach p = (s, ..., t) in P(e), где feasible(p) = true do
4          links(p) ← links(p) − 1
5          if connected(s, t) = false and links(p) = 0 then
6              connected(s, t) ← true
7              connectedPairs ← connectedPairs + 1
8              if connectedPairs = k×(k−1)/2 then
9                  RL ← RL + P_l * 1
10                 RU ← RU − P_l * (1 − 1)
11                 return
12             end
13         end
14     end
15     FACTO(Input, P_l)
16 end
```

# C&P, уточняющий оценки

```
1  Function deleteEdge(Input, e, P_l)
2      P_l ← P_l * r_e
3      foreach p = (s, ..., t) in P(e), где feasible(p) = true do
4          feasible(p) ← false
5          np(s, t) ← np(s, t) − 1
6          if np(s, t) = 0 then
7              RL ← RL + P_l * 0
8              RU ← RU − P_l * (1 − 0)
9          end
10     end
11     FACTO(Input, P_l)
12 end
```

# Публикации


- Мигов Д.А., Нестеров С.Н., Родионов А.С. Методы ускорения расчета надежности сетей с ограничением на диаметр. Вестник СибГУТИ. № 1, 2014, с. 49-56.
- S. Nesterov, D. Migov. Parallel method for reliability calculation of diameter constrained networks. Bulletin of the Novosibirsk Computing Center. Series: Numerical Analysis, issue 17, 2015.
- D. Migov, S. Nesterov. Methods of Speeding up of Diameter Constrained Network Reliability Calculation. Springer Lecture Notes in Computer Science (in ICCSA 2015). 2015.