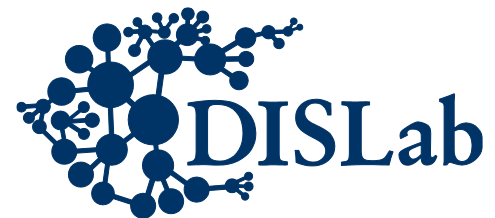


Параллельная обработка больших графов

Занятие 7

А.С. Семенов

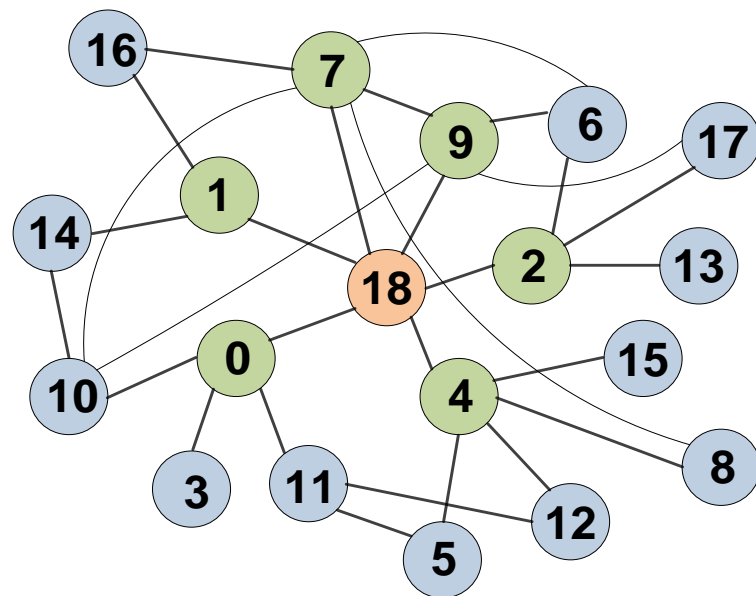
dislab.org



Проблемы и подходы к решению задач обработки графов в рамках одного вычислительного узла

Поиск вширь в графе

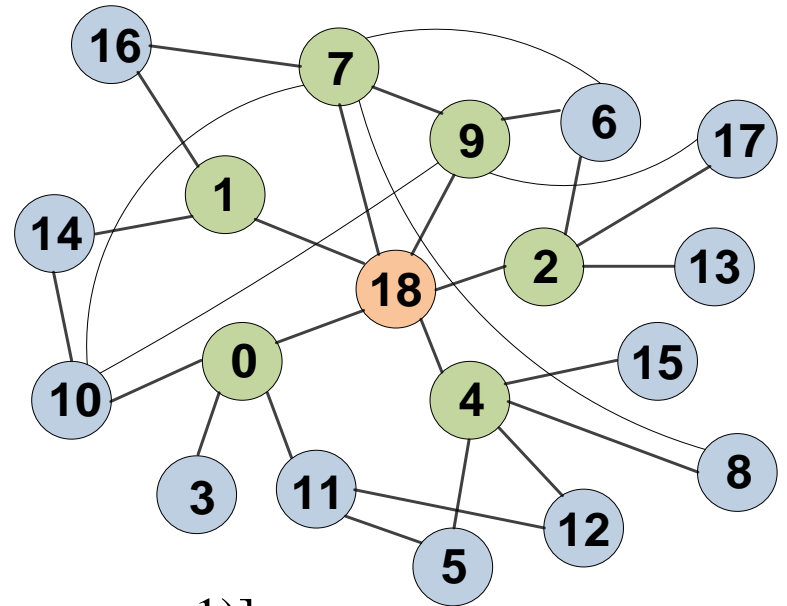
```
Q = {v_start}
Visited = {v_start}
while Q ≠ {}
  Q_next = {}
  for all vertex ∈ Q do
    for all w: (vertex, w) ∈ E do
      if w ∉ Visited then
        Q_next = Q_next ∪ w
        Visited = Visited ∪ w
      endif
    end for
  end for
  Q = Q_next
end while
```



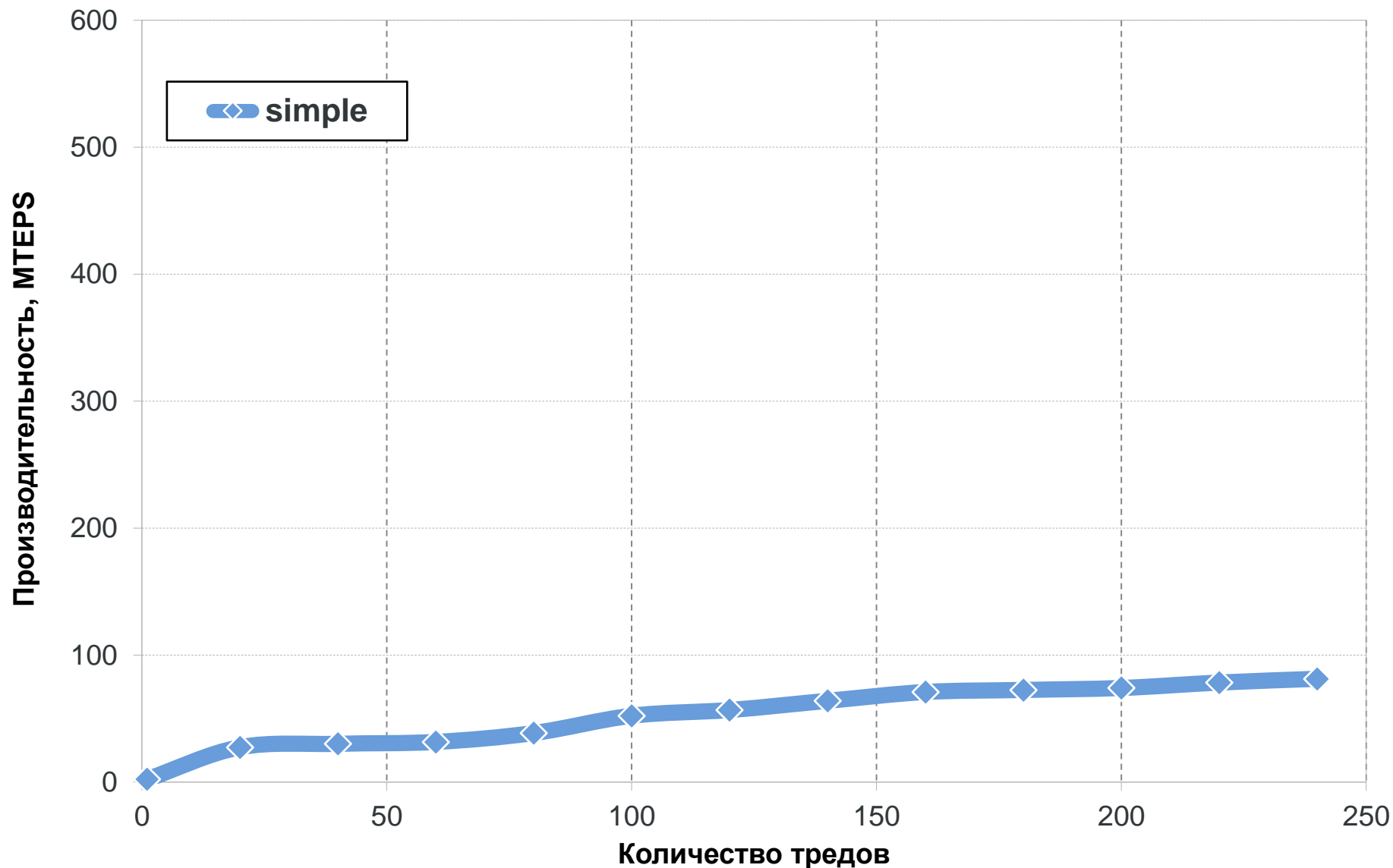
Поиск в ширь в графе (BFS)

Подход Queue-based, алгоритм simple

```
Q_counter = 1
Q[0] = root
Visited[root] = 1
while Q_counter > 0
    Q_next_counter = 0
    #pragma omp parallel for
    for all vertex ∈ Q do
        for all w: (vertex, w) ∈ E do
            if Visited[w] == 0 then
                Q_next[__sync_fetch_and_add(Q_next_counter, 1)] = w
                Visited[w] = 1
            endif
        end for
    end for
    swap(Q, Q_next) // обмен Q и Q_next
end while
```

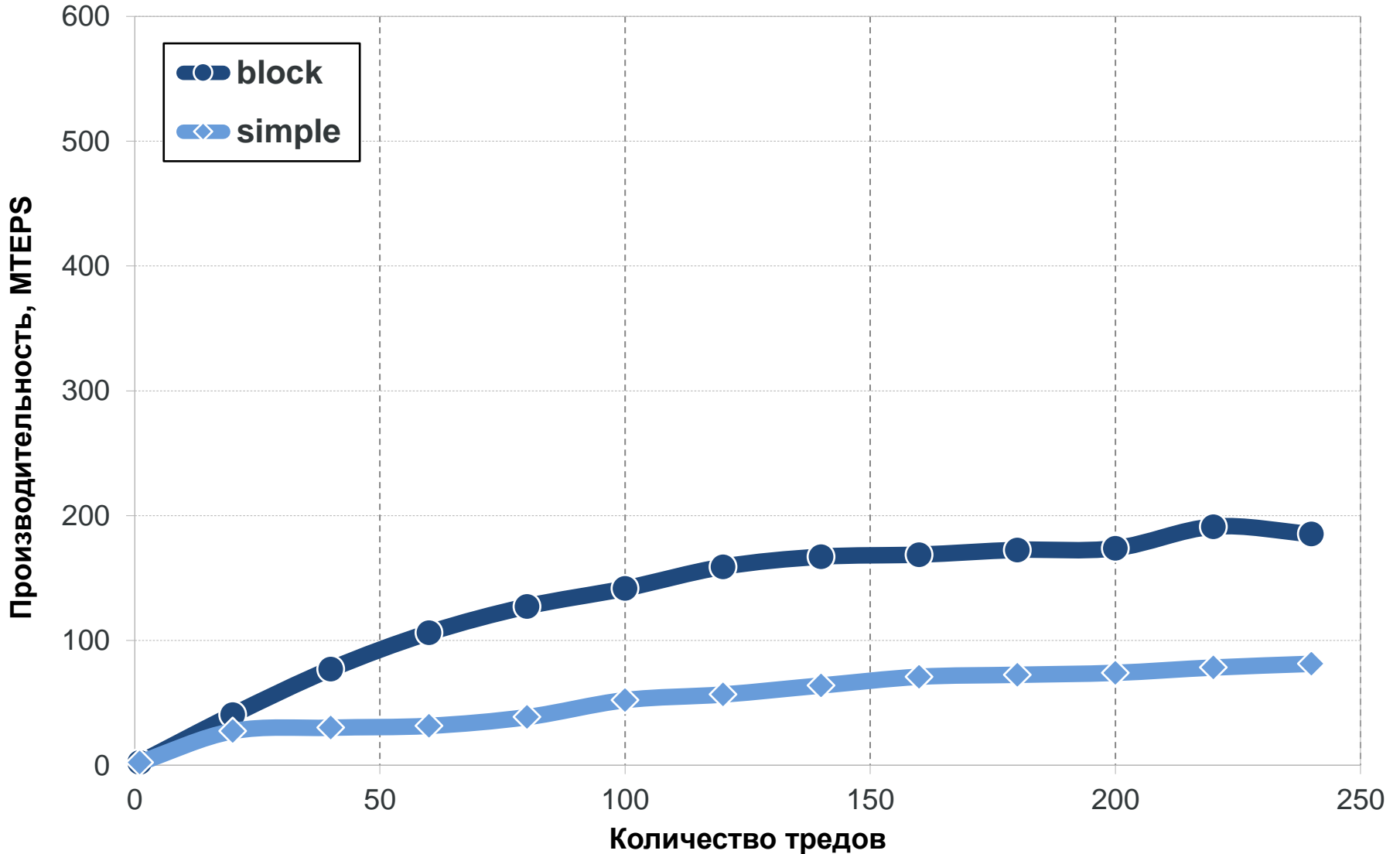


Производительность алгоритма simple в зависимости от числа используемых тредов на сопроцессоре Phi-5110P



Число вершин в графе: $N = 2^{27}$ (134 млн), средняя связность вершины: $k = 8$

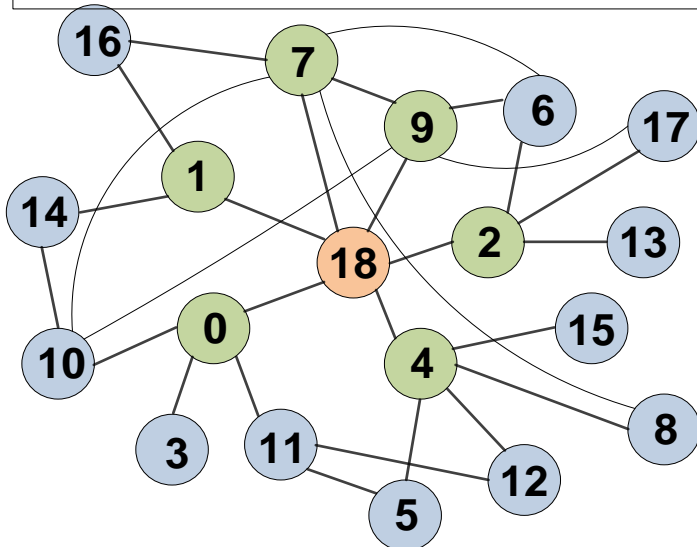
Производительность алгоритмов simple и block в зависимости от числа используемых тредов на сопроцессоре Phi-5110P



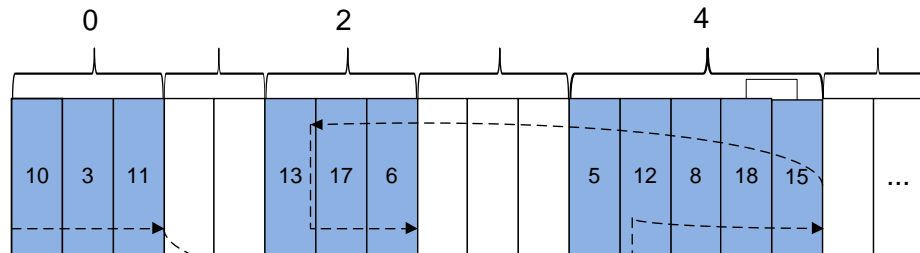
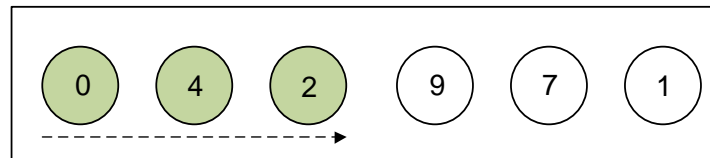
Число вершин в графе: $N = 2^{27}$ (134 млн), средняя связность вершины: $k = 8$

Недостатки подхода Queue-based

```
#pragma omp parallel for
for all vertex  $\in$  Q do
  for all w: (vertex, w)  $\in$  E do
    if Visited[w] == 0 then
      Qnext[__sync_fetch_and_add(Qnext_counter, 1)] = w
      Visited[w] = 1
    endif
  end for
end for
```

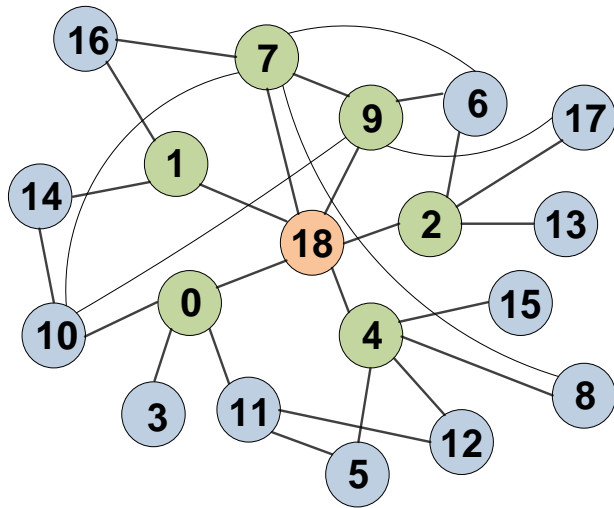


массив Q



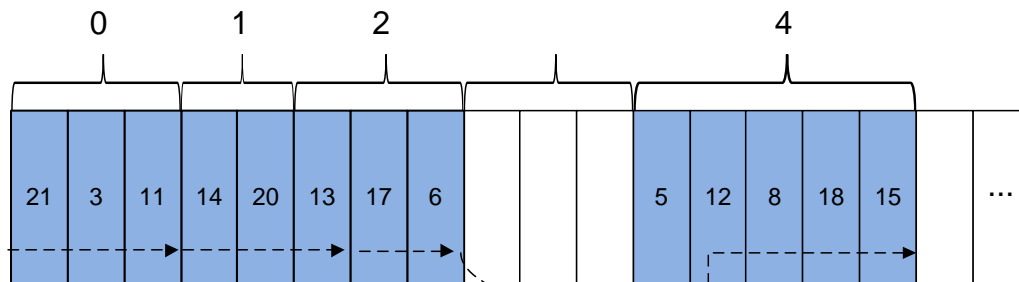
массив смежных вершин

Подход Read-based, алгоритм read



массив levels

| 0 | 1 | 2 | 3 | 4 | | | | | |
|---|---|---|-----|---|-----|-----|---|---|---|
| 1 | 1 | 1 | INF | 1 | INF | INF | 1 | 0 | 1 |

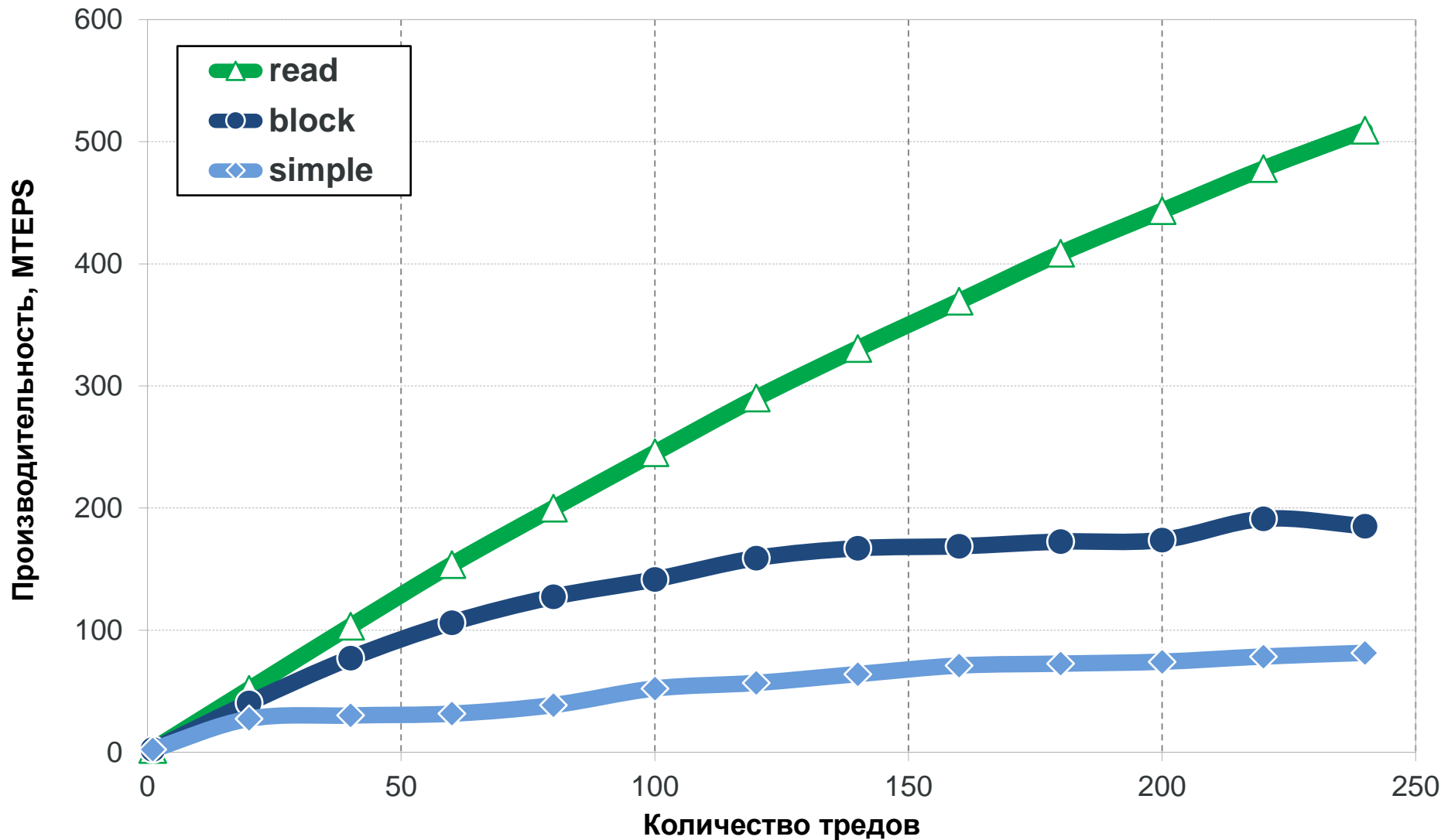


массив смежных вершин

```

#pragma omp parallel for reduction (...)
for all vertex  $\in V$  do
  if levels[vertex]  $\neq$  numLevel then
    continue
  for all w: (vertex, w)  $\in E$  do
    if levels[w] == -1 then
      levels[w] = numLevel + 1
      nLevelVerts = nLevelVerts + 1
    end if
  end for
end for
  
```


Производительность алгоритмов simple, block и read в зависимости от числа используемых тредов на сопроцессоре Phi-5110P



Число вершин в графе: $N = 2^{27}$ (134 млн), средняя связность вершины: $k = 8$

Алгоритм bottom-up-hybrid

```
#pragma omp parallel for reduction (...)
```

```
for all vertex  $\in V$  do
```

```
  if levels[vertex] == -1 then
```

```
    for all  $w: (vertex, w) \in E$  do
```

```
      if levels[w] == numLevel then
```

```
        levels[vertex] = numLevel + 1
```

```
        nLevelVerts = nLevelVerts + 1
```

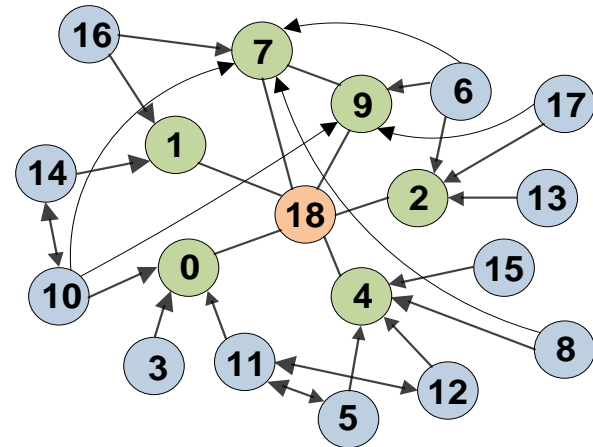
```
      break
```

```
    end if
```

```
  end for
```

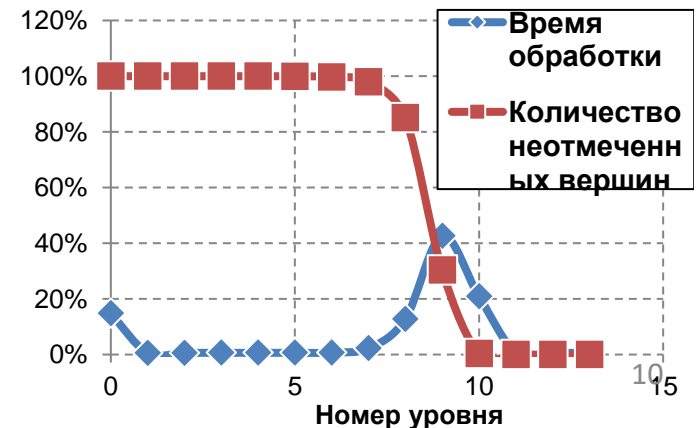
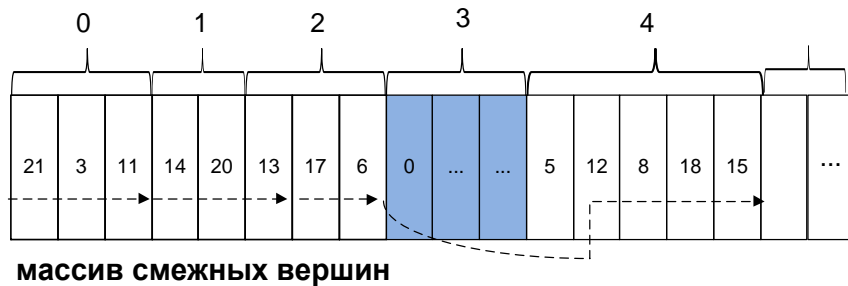
```
end if
```

```
end for
```

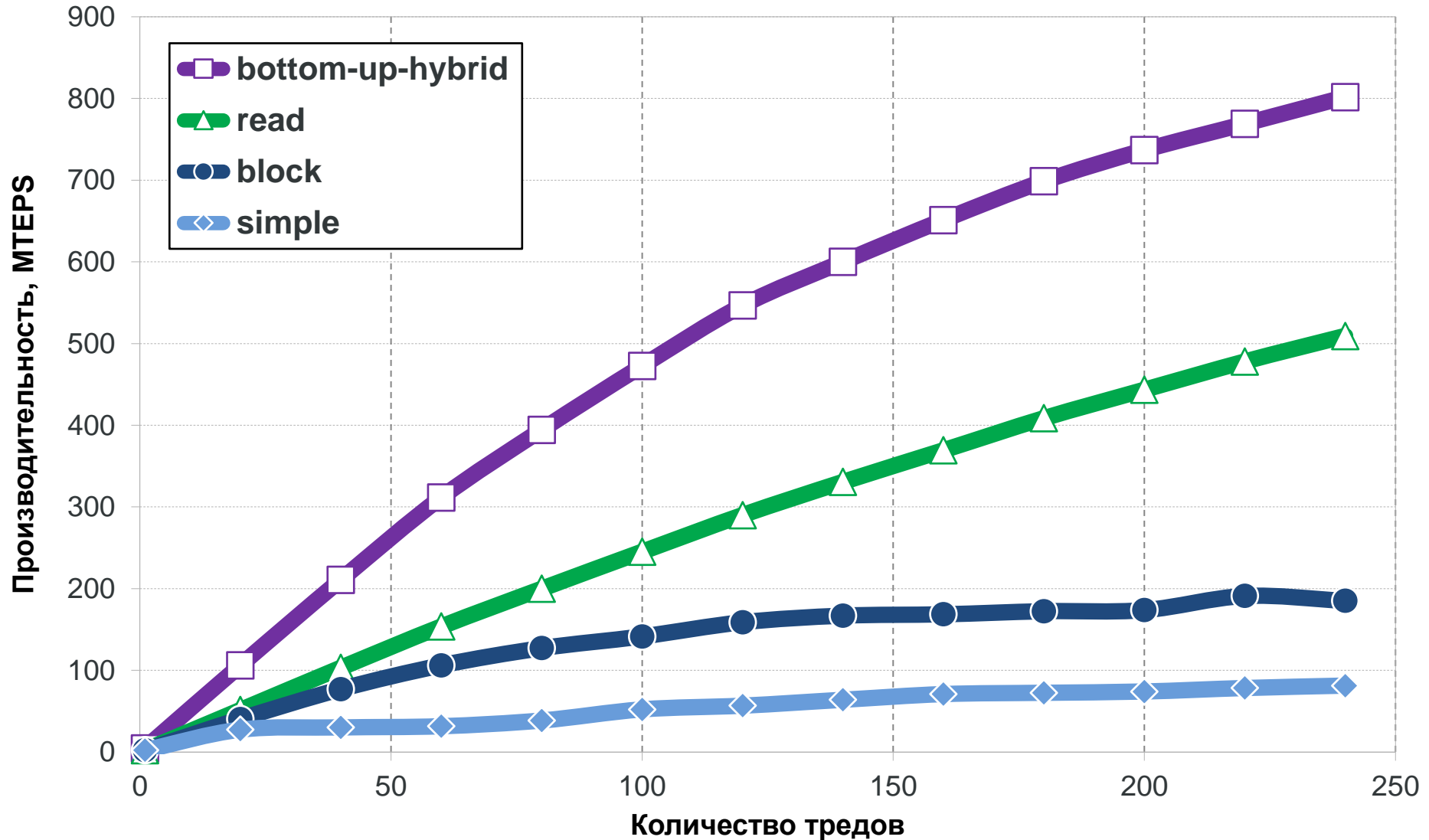


массив levels

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|-----|---|-----|-----|---|---|---|----|
| 1 | 1 | 1 | INF | 1 | INF | INF | 1 | 0 | 1 | |

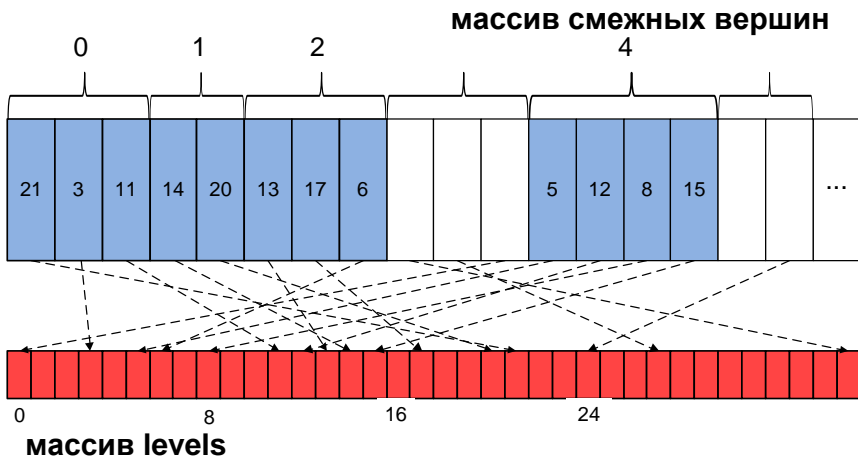
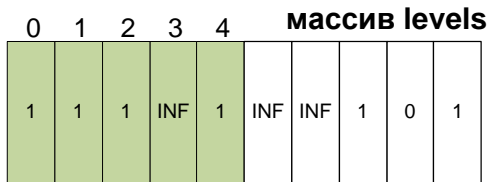
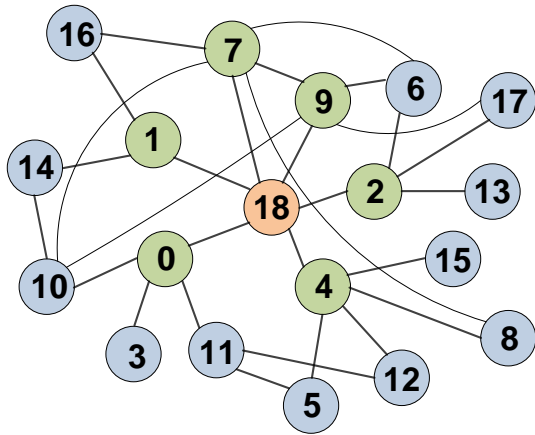


Производительность алгоритмов simple, block, read и bottom-up-hybrid в зависимости от числа используемых тредов на сопроцессоре Phi-5110P



Число вершин в графе: $N = 2^{27}$ (134 млн), средняя связность вершины: $k = 8$

Недостатки алгоритмов read и bottom-up-hybrid

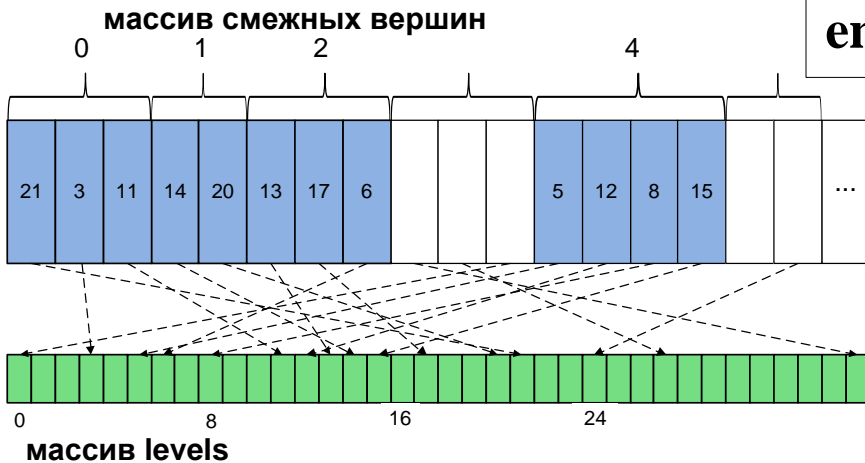
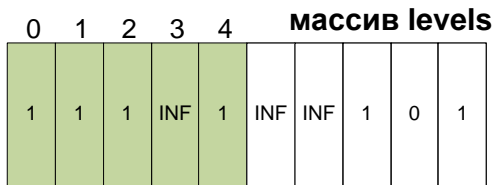
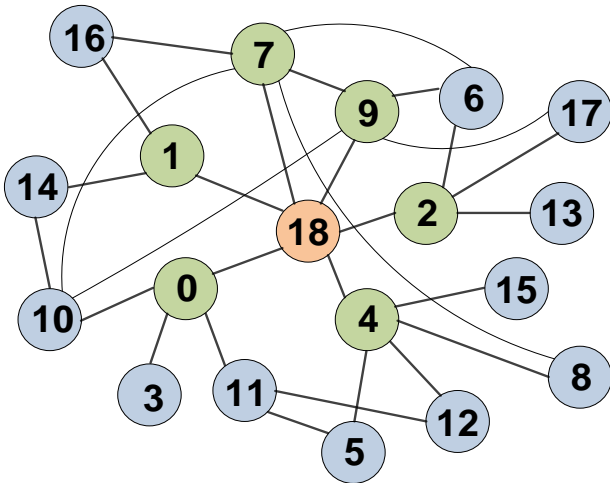


```

#pragma omp parallel for reduction
(...)
for all vertex ∈ V do
  if levels[vertex] ≠ numLevel then
    continue
  for all w: (vertex, w) ∈ E do
    if levels[w] == -1 then
      levels[w] = numLevel + 1
    ...
  end if
end for
end for
    
```

| | SB | Phi-5110P |
|-------------------------------------|------|-----------|
| Частота, ГГц | 2.2 | 1.05 |
| Задержка обращения в память (такты) | ~150 | ~300 |

Решение: ручная развертка цикла + использование prefetch

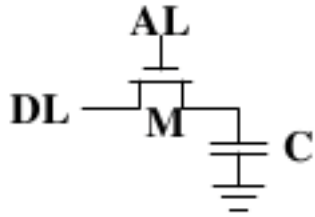


```
#pragma omp parallel for reduction (...)
for all vertex ∈ V do
  if levels[vertex] ≠ numLevel then continue
  for all w: (vertex, w) ∈ E do
    prefetch(levels[w])
    ...
    if levels[w] == -1 then
      levels[w] = numLevel + 1
    ...
  end if
end for
end for
```

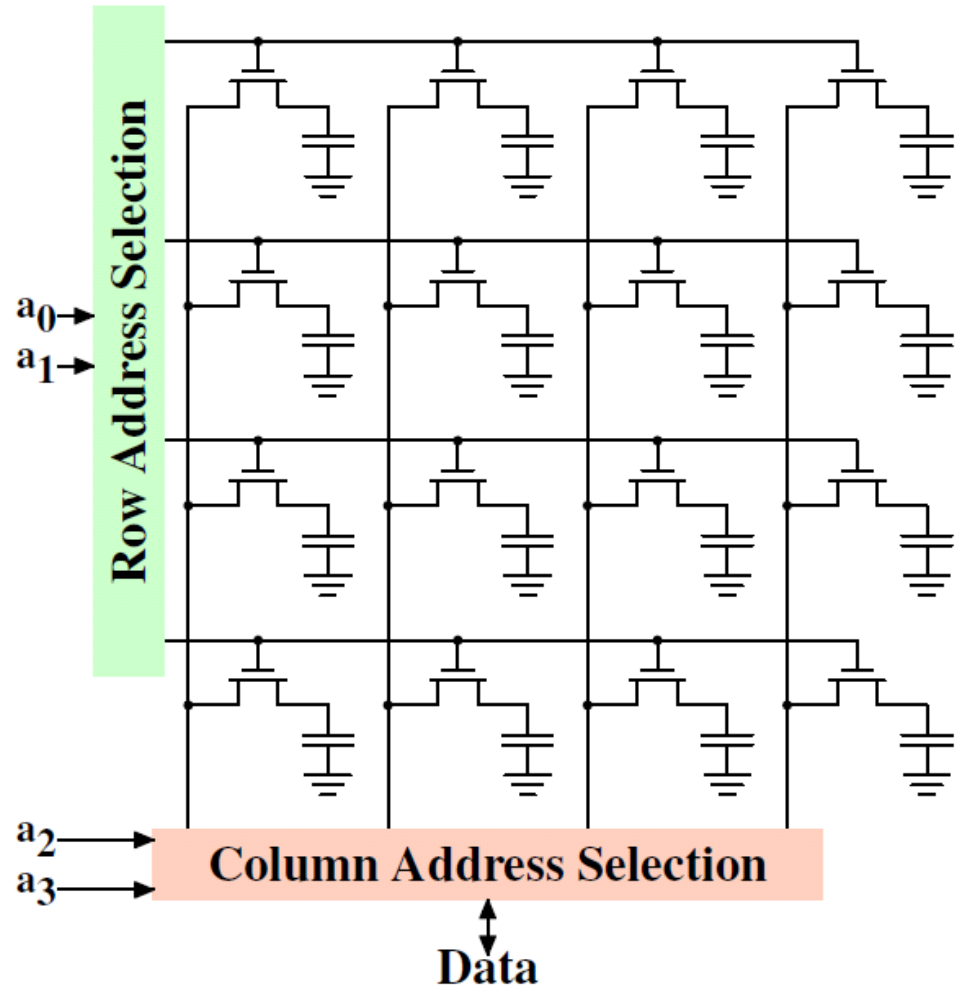
| | SB | Phi-5110P |
|--|-------------|-----------|
| Пиковая ПС памяти, ГБ/с | 51 | 352 |
| ПС чтения из памяти, ГБ/с; Последовательный / случайный доступ | 42 / 3.3 | 183 / 3.8 |
| Задержка, тактов | 200 | 300 |

Память SDRAM

- Память организована как матрица

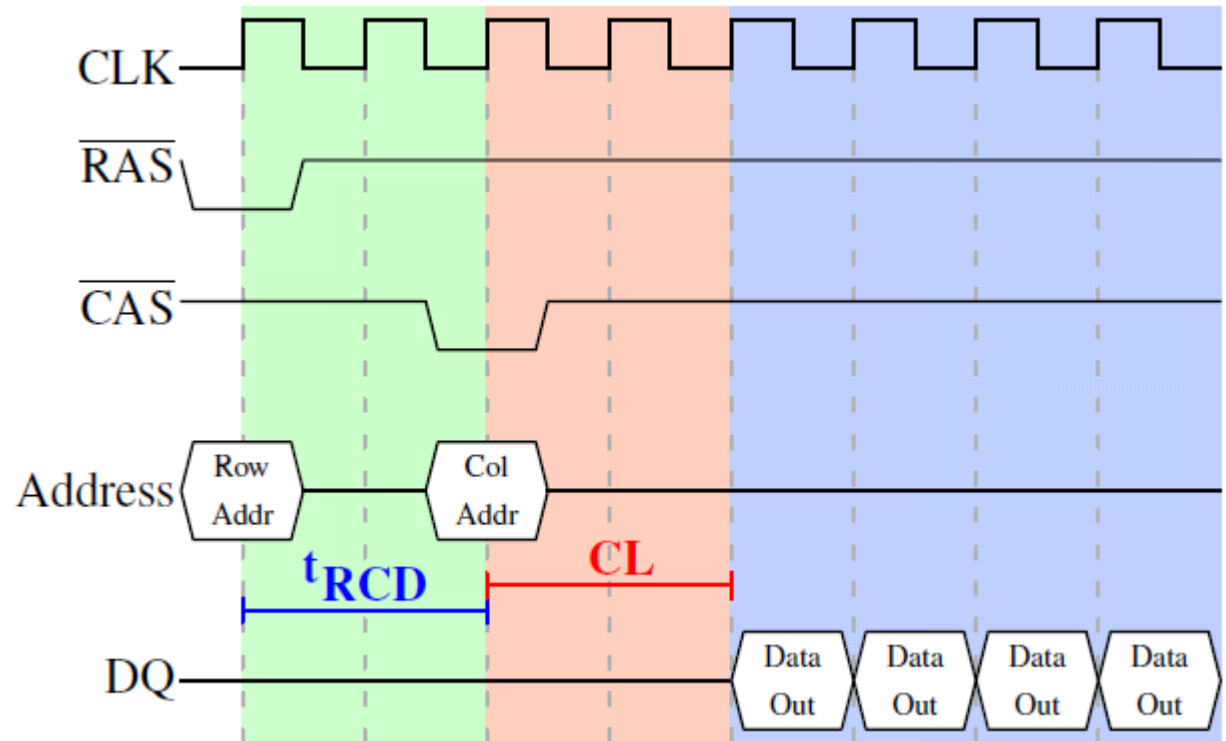


- На все операции требуется энергия и время
- Чтение
 - Усилитель считывания
 - Перезарядка после чтения (энергия и время)
- Перезарядка памяти



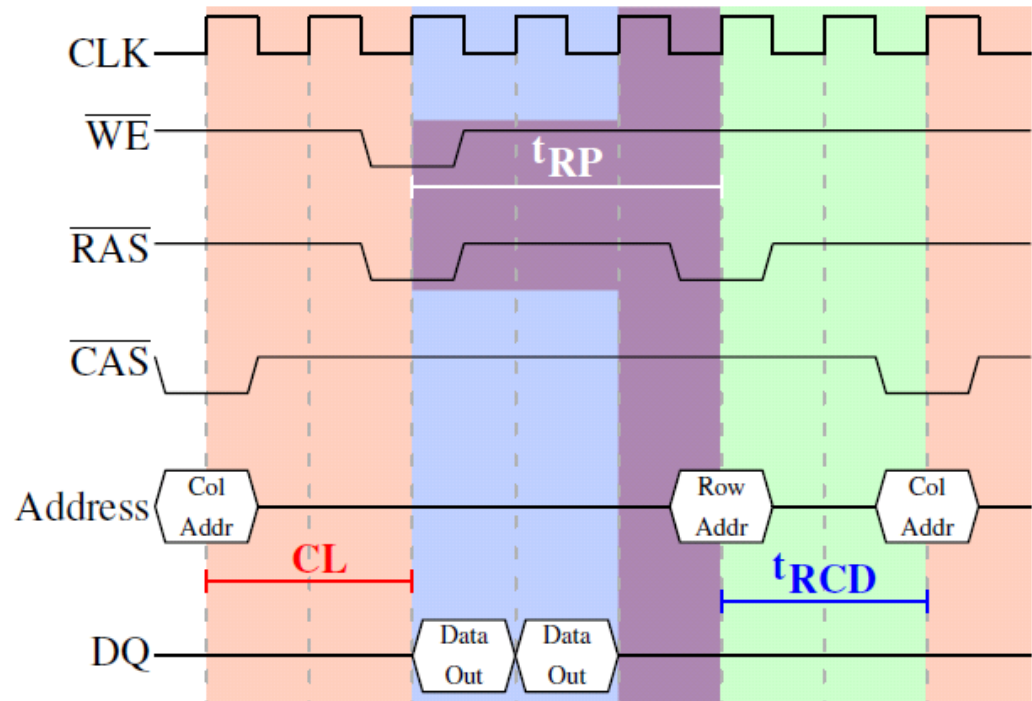
Память SDRAM

- На определение состояния и перезарядку требуется время

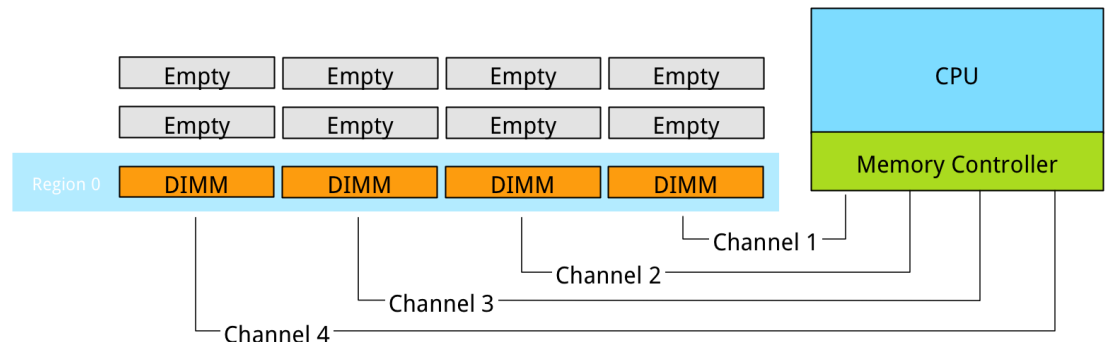


Память SDRAM

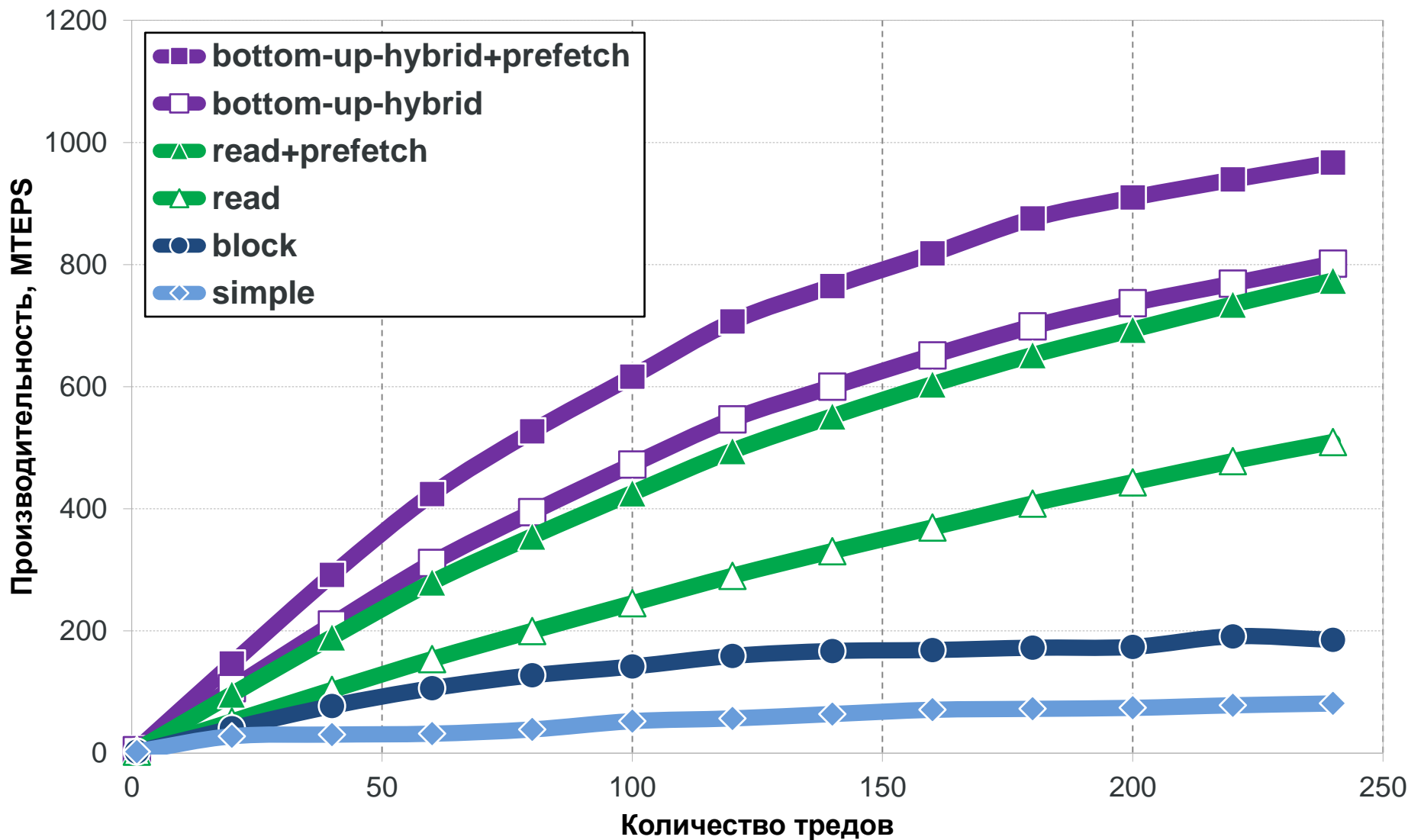
- Чтение памяти, необходимо подзаряжать конденсаторы
- Необходимость перезарядки конденсаторов (токи утечки)
- t_{RP} - время предварительной зарядки
- Каждая строка должна быть перезаряжена каждые 7.8 мкс



Иерархия памяти:
контроллер – канал – банк

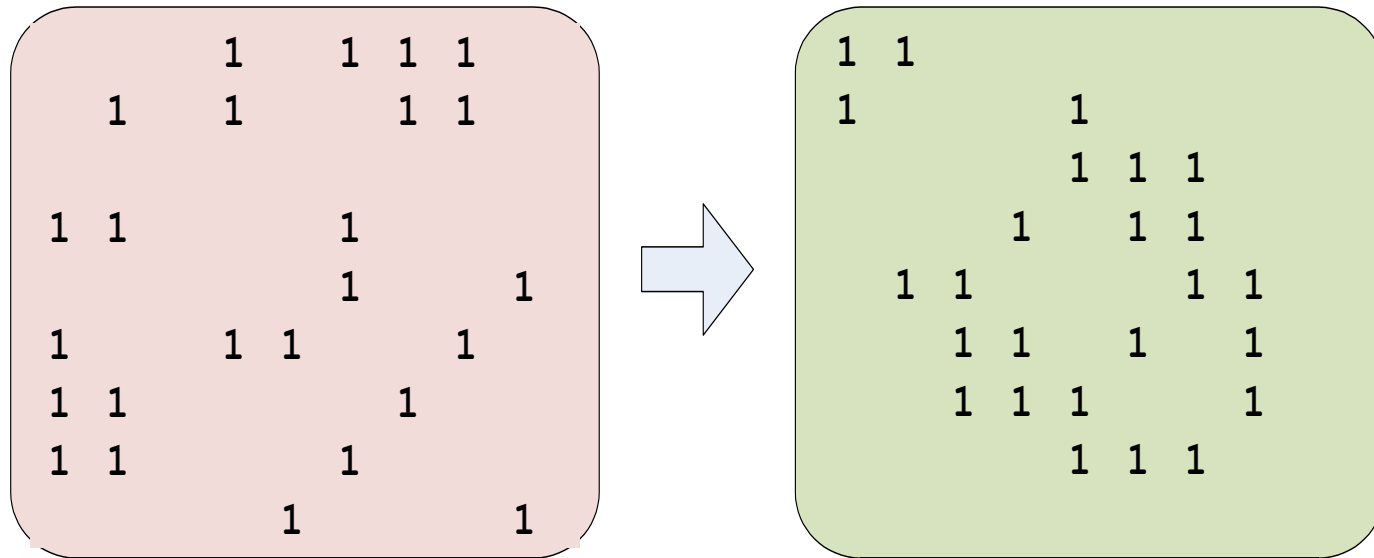


Производительность алгоритмов simple, block, read и bottom-up-hybrid с префетчем в зависимости от числа используемых тредов на сопроцессоре Phi-5110P



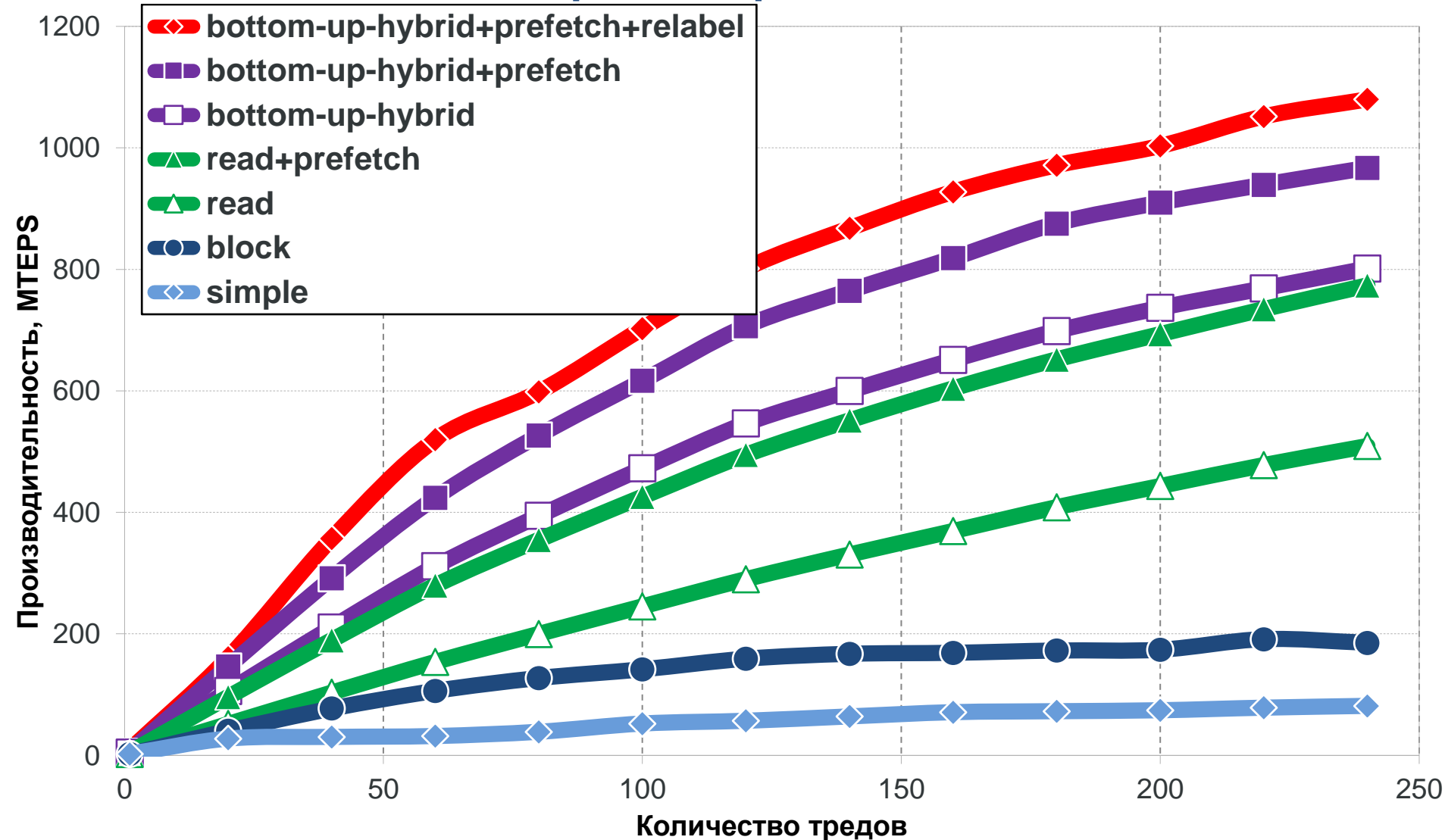
Число вершин в графе: $N = 2^{27}$ (134 млн), средняя связность вершины: $k = 8$

Улучшение локализации: перестановка вершин



- Матрица смежности приводится к ленточному виду с уменьшением ширины ленты (алгоритм Reverse Cuthill-McKee) => уменьшается количество кэш-промахов
- Списки смежных вершин сортируются => уменьшается количество промахов в TLB
- Использование больших страниц

Производительность различных алгоритмов, с префетчем и перестановками в зависимости от числа используемых тредов на сопроцессоре Phi-5110P

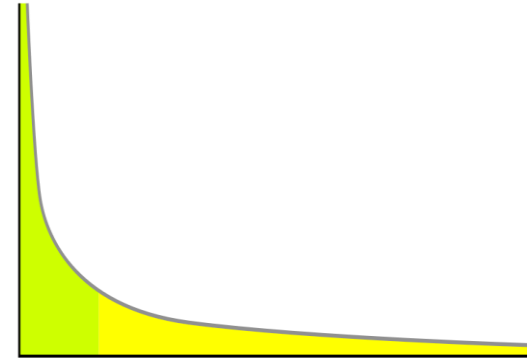


Число вершин в графе: $N = 2^{27}$ (134 млн), средняя связность вершины: $k = 8$

Распараллеливание: дисбаланс вычислительной нагрузки

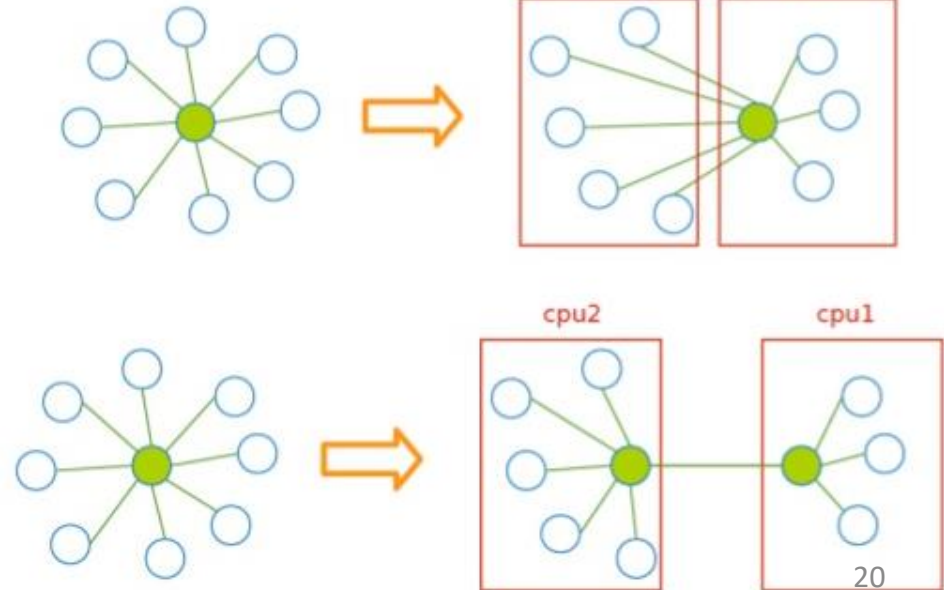
- **Проблема: неравномерность** итераций циклов

```
# pragma omp parallel for
for (int u = 0; u < G->n; u++)
    for (int j = G->rowsIndices[u]; j < rowsIndices[u+1]; j++) {
        .....
    }
```



- **Решение 1:** #pragma omp parallel for **schedule (guided)** – для **динамического** распределения вершин по тreads

- **Решение 2:** На этапе предобработки выполнение процедуры Vertex-cut: разделение вершины и **разрезание** списков смежности вершин



Большой объем памяти

- **Проблема:** постоянная смена данных в кэше, низкие характеристики при случайном доступе
- **Решения** на этапе предобработки:
 - Хранение только половины графа (для неориентированного)
 - Удаление кратных ребер
 - Перестановка вершин (Cuthill-McKee)
 - Сжатие данных
 - `edge_id_t: uint64_t --> uint32_t`
 - Сортировка ребер каждой вершины
 - Сортировка всех ребер графа

Резюме: проблемы и подходы к решению задач в рамках одного узла

- Выбор оптимального представления графа
- По возможности организация последовательного доступа к данным
- По возможности избегать использовать межпоточковые синхронизации
- Стремиться работать не на задержке обращений к памяти, а на темпе
- Улучшение локализации
- Алгоритмические оптимизации
- Сжатие данных
- Аккуратная работа с памятью внутри NUMA-вычислительного узла
- Балансировка нагрузки
- Аккуратно измерять производительность