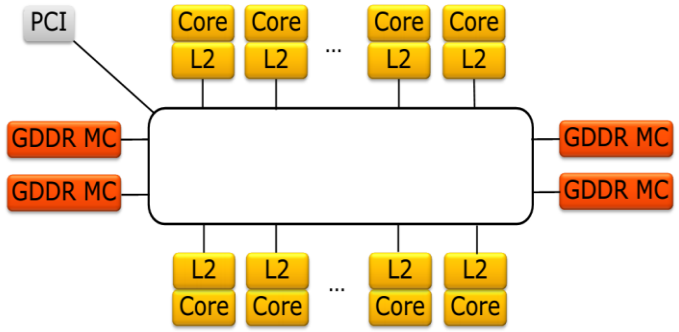
Научный семинар по информационным технологиям.
г. Челябинск, 20 октября 2015 г.

Параллельный алгоритм кластеризации Partitioning Around Medoids для многоядерного сопроцессора Intel Xeon Phi

Т.В. Речкалов
Научный руководитель: **М.Л. Цымблер**

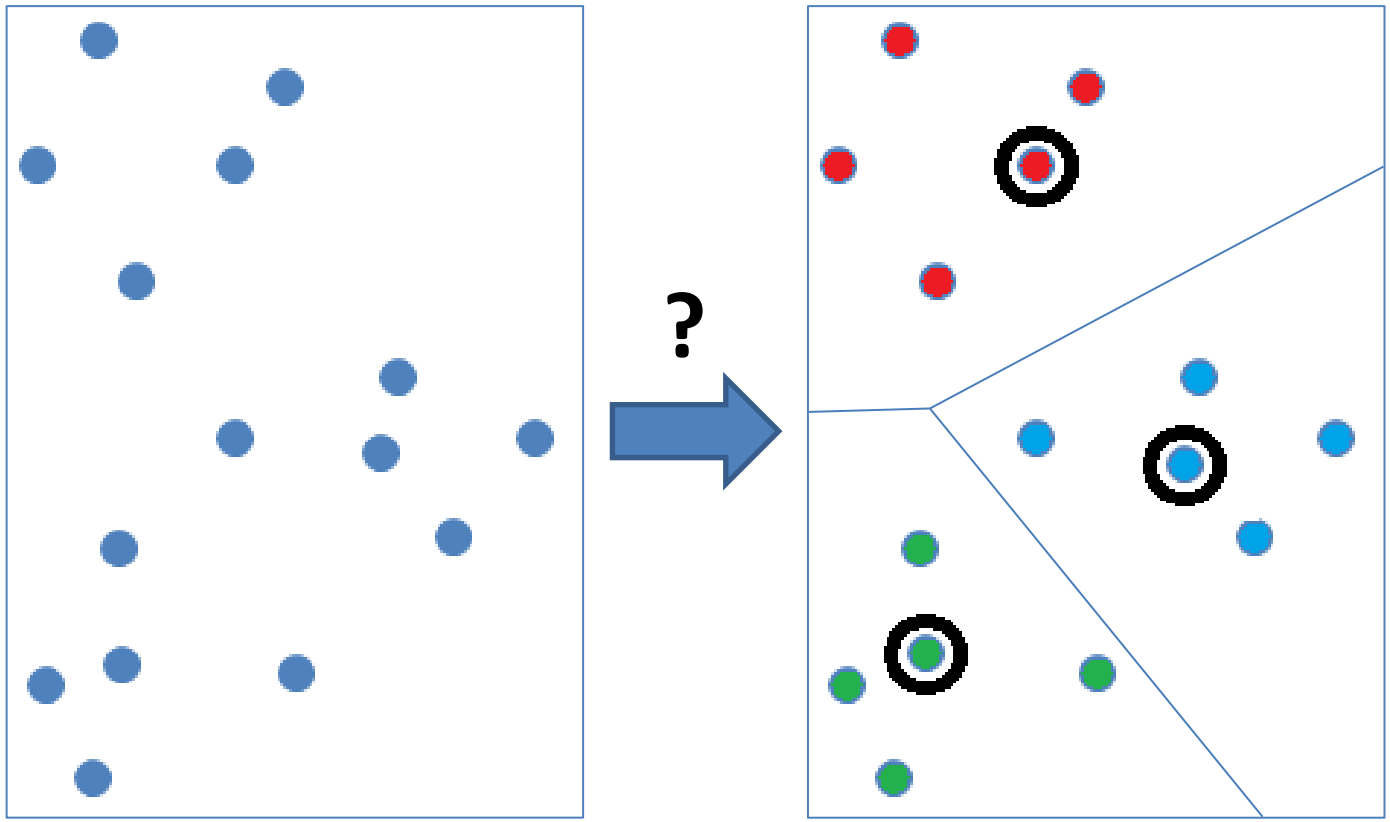
Intel Xeon Phi

- Общее с x86
  - набор инструкций
  - модель программирования
  - модель распараллеливания



- Отличия от x86
  - большое количество ядер (>60)
  - большие векторные регистры (512 бит)

Задача кластеризации



Характеристика алгоритма PAM

- Алгоритм *PAM (Partitioning Around Medoids)* - разделительный алгоритм кластеризации, в котором в качестве центров кластеров выбираются только кластеризуемые объекты (называемые *медоидами*)
- Временная сложность итерации $O(k(n-k)^2)$, где
  - n – количество кластеризуемых объектов
  - k – количество искомых кластеров

Стоимостная функция

- Стоимостная функция

$$E = \sum_{j=1}^{n} \min_{1 \le i \le k} \rho(c_i, o_j).$$

, где

c_i – медоид,

o_j – кластеризуемый объект,

ρ – функция расстояния

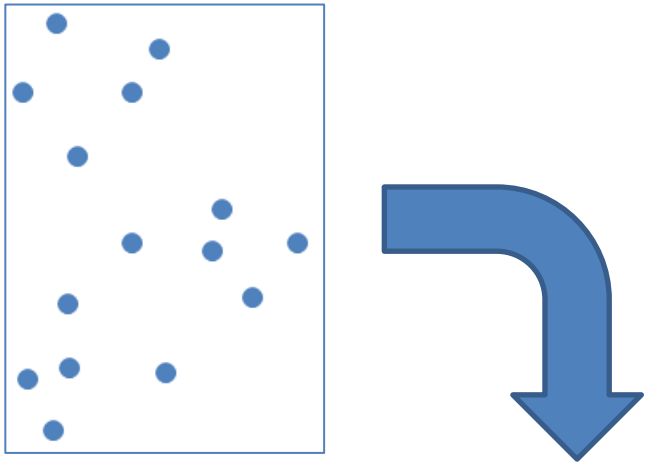
Псевдокод PAM

Вход : Множество объектов *O*, количество кластеров *k*

Выход: Множество кластеров *C*

1. Инициализировать *C*; // фаза BUILD
2. **repeat** // фаза SWAP
3. Найти лучшую оценку замены T_{min};
4. Поменять местами c_{min} и o_{min}, определяющих T_{min};
5. **until** $T_{min} < 0$;

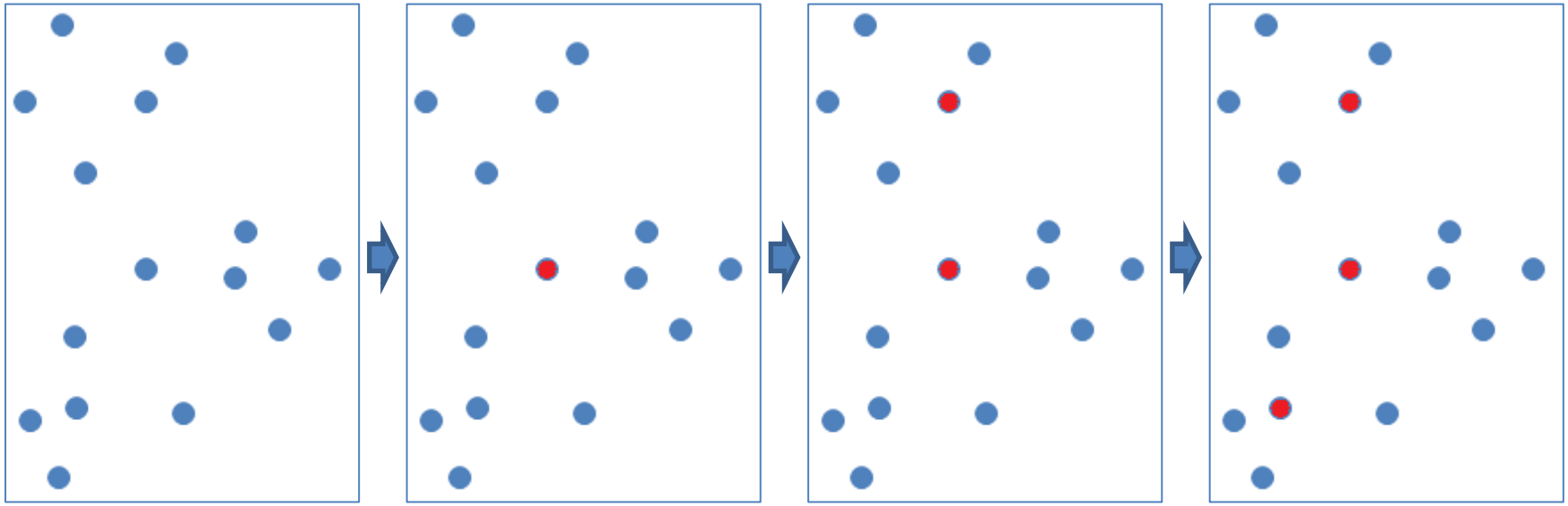
Вычисление матрицы расстояний



	o_1	o_2	o_3	…	o_n
o_1	$\rho(o_1,o_1)$	$\rho(o_1,o_2)$	$\rho(o_1,o_3)$	…	$\rho(o_1,o_n)$
o_2	$\rho(o_2,o_1)$	$\rho(o_2,o_2)$	$\rho(o_2,o_3)$	…	$\rho(o_2,o_n)$
o_3	$\rho(o_3,o_1)$	$\rho(o_3,o_2)$	$\rho(o_3,o_3)$	…	$\rho(o_3,o_n)$
…	…	…	…	…	…
o_n	$\rho(o_n,o_1)$	$\rho(o_n,o_2)$	$\rho(o_n,o_3)$	…	$\rho(o_n,o_n)$

Фаза BUILD

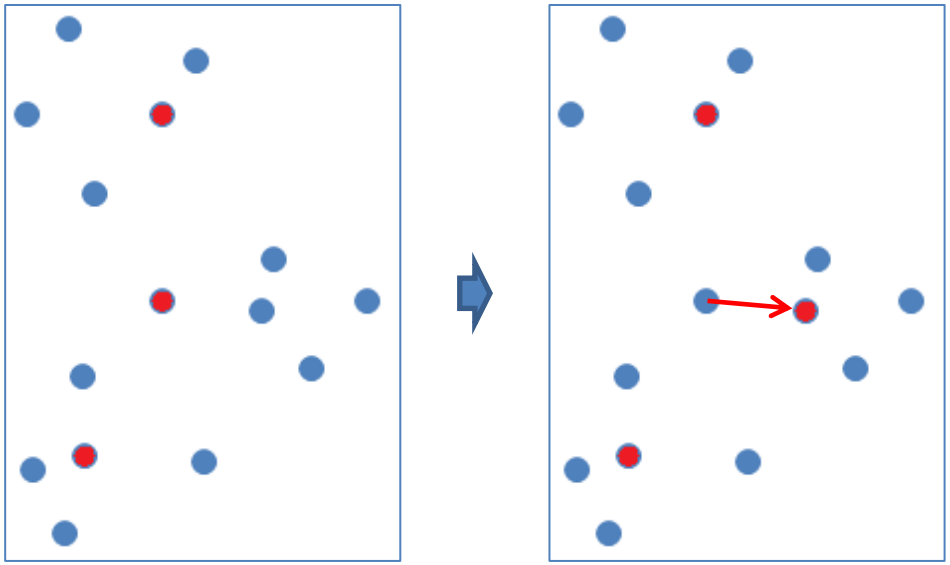
k=3



E=∞ *E*= 2.250 *E*= 1.578 *E*= 1.014

Временная сложность $O(kn^2)$

Фаза SWAP



$E = 1.014$ $E = 0.865$

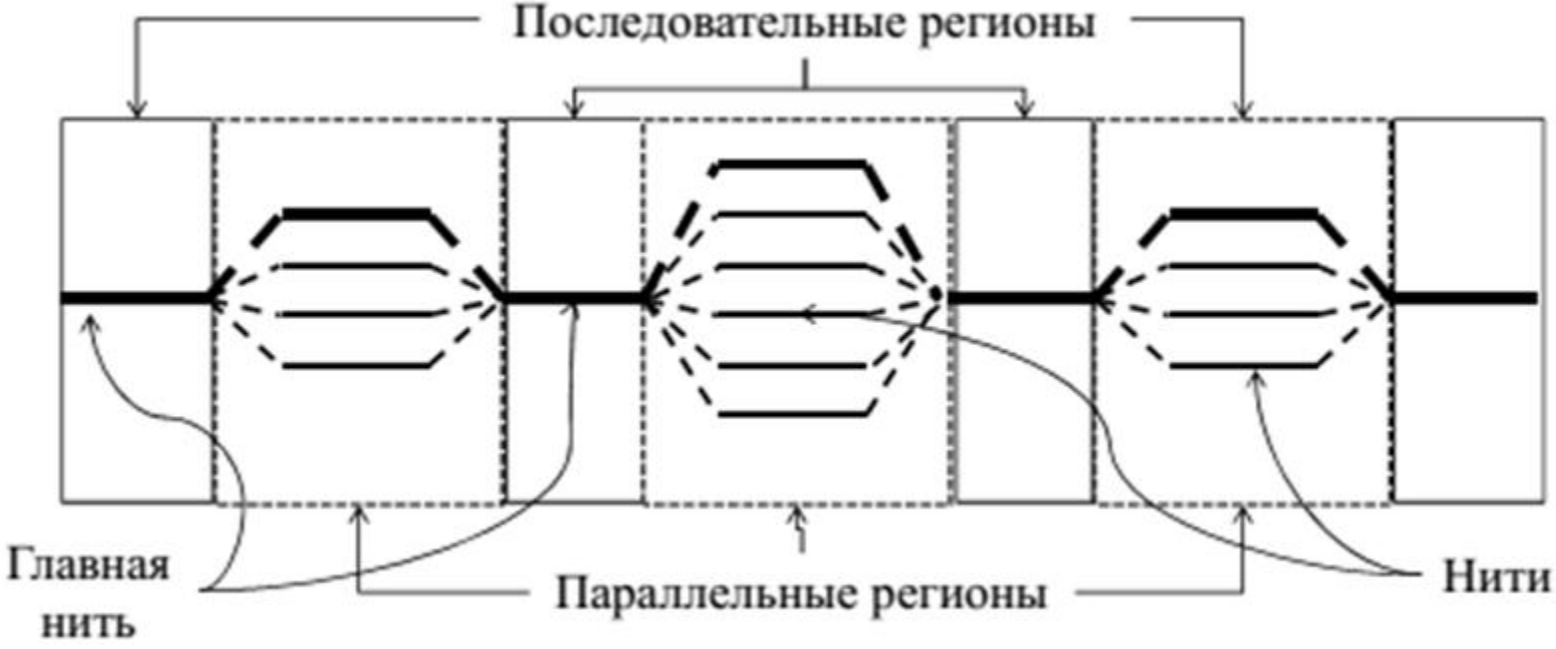
Временная сложность $O(k(n - k)^2)$ на итерацию

Используемые оптимизации

- OpenMP
- Реорганизация циклов
  - Данные разбиты на блоки по 32 элемента
- Тайлинг

OpenMP

- Главная нить(программа) порождает семейство дочерних нитей(сколько необходимо). Порождение и завершение осуществляется с помощью директив компилятора.



Реорганизация циклов

- SIMD — Single Instruction Multiple Data
- Векторные регистры 512 бит
  - 16 float
  - 8 double

Скалярный цикл

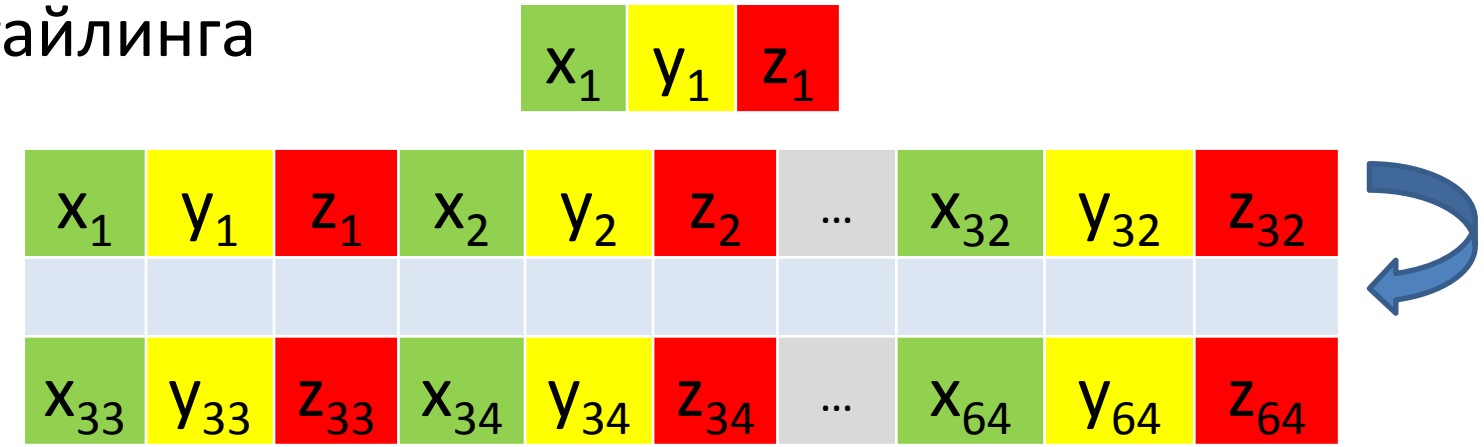
```
for(i = 0; i < n; ++i)
    a[i] = b[i] + c[i];
```

SIMD цикл

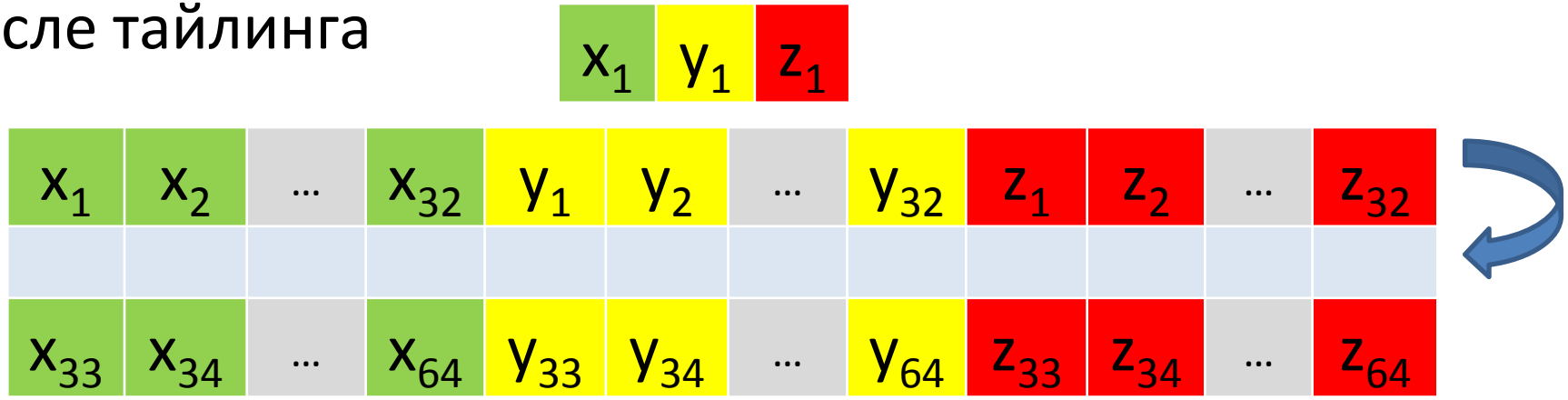
```
#ifdef DOUBLE_PRECISION
  #define LEN 8
#else
  #define LEN 16
#endif
for(i = 0; i < n; i += LEN)
    a[i:LEN] = b[i:LEN] + c[i:LEN];
```

Тайлинг

Без тайлинга



После тайлинга



Реализация PAM

Вход : Множество объектов *O*, количество кластеров *k*

Выход: Множество кластеров *C*

1. M <- PrepareDistanceMatrix(O);
2. C <- BuildMedoids(M); // фаза BUILD
3. **repeat** // фаза SWAP
4. T_{min} <- FindBestSwap(M, C);
5. Поменять местами c_{min} и o_{min}, определяющие T_{min};
6. **until** $T_{min} < 0$;

PrepareDistanceMatrix

Вход : Множество объектов O
Выход: Матрица расстояний M

```
1  forall the o_i таких, что 1 ≤ i ≤ n do // parallelized
2      for j = 1 до n шаг L do
3          for k = 1 to p do
4              for l такое, что j ≤ l ≤ j + L do // vectorized
                   // доступ к o_l организован с применением тайлинга
5                  m_il ← m_il + (o_i[k] − o_l[k])^2;
6              end
7          end
8          for l такое, что j ≤ l ≤ j + L do // vectorized
9              m_il ← sqrt(m_il);
10         end
11     end
12 end
```

Вычисление матрицы расстояний

```
void prepareDistanceMatrix(const float* rowData, const float* colData,
        float* distances, const int n, const int pointWidth){
  const int vecLen = 32;
  #pragma omp parallel
  {
    float point[pointWidth] __attribute__((aligned(64)));
    float result[vecLen]    __attribute__((aligned(64)));
    #pragma omp for
    for(int i=0; i<n; ++i){
      point[:]=rowData[i*pointWidth:pointWidth];
      for(int ii = 0; ii < n; ii += vecLen){
        result[:]=0;
        for(int j=0; j < pointWidth; ++j){
          const float* restrict point2 = colData+ii*pointWidth;
          result[:]+=(point[j]-point2[j*vecLen:vecLen])*
                     (point[j]-point2[j*vecLen:vecLen]);
        }
        distances[i*n+ii:vecLen] = sqrtf(result[:]);
      }
    }
  }
}
```

	OpenMP		Тайлинг
	Реорганизация циклов		Выравнивание

BuildMedoids

Вход : Матрица расстояний M
Выход: Множество медоидов C

```
1  forall the i = 1 to n do // parallelized
2      if sum_{j=1}^{n} m_ij минимальна then // вычисление суммы векторизовано
3          c_1 ← o_i;
4      end
5  end
6  Инициализировать вектор расстояний до ближайшего медоида D
7  for l = 2 to k do
8      forall the i = 1 to n do // parallelized
           // вычисление суммы векторизовано
9          if sum_{j=1}^{n} min(d_j, m_ij) минимальна then
10             c_l ← o_i;
11         end
12     end
13     Обновить D;
14 end
```

FindBestSwap

Вход : Матрица расстояний M, множество медоидов C
Выход: T_{min}

1 Инициализировать массив величин переключения объектов T
2 **forall the** o_h *таких, что* $1 \leq h \leq n$ *и* o_h *не медоид* **do** `// parallelized`
3 | **for** $l = 1$ *до* n *шаг* L **do**
4 | | **for** $i = 1$ *до* k **do**
5 | | | $T_{ih} \leftarrow T_{ih} + \sum_{j=l}^{l+L} C_{jih}$;
6 | | **end**
7 | **end**
8 **end**
9 $T_{min} \leftarrow \min_{1 \leq h \leq n, 1 \leq i \leq k} T_{ih}$;

Особенности низкого уровня

- C_{jih} — это вклад не-медоида o_j в изменение стоимостной функции при замене медоида c_i на не-медоид o_h

Вход : o_j, c_i, o_h, d_j, s_j
Выход: C_{jih}
if $\rho(o_j, c_i) > d_j$ **and** $\rho(o_j, o_h) > d_j$ **then**
  $C_{jih} \leftarrow 0$
else if $\rho(o_j, c_i) = d_j$ **then**
  if $\rho(o_j, o_h) < s_j$ **then**
    $C_{jih} \leftarrow \rho(o_j, o_h) - d_j$
  else
    $C_{jih} \leftarrow s_j - d_j$
  end
else if $\rho(o_j, o_h) < d_j$ **then**
  $C_{jih} \leftarrow \rho(o_j, o_h) - d_j$
end

PAM-1 и PAM-2

- Вычисление C_{jih} плохо векторизуемо, поэтому было реализовано две версии алгоритма PAM
- В PAM-1 применяется «принудительная» векторизация
  - #pragma vector always
- В PAM-2 кэшируется больше данных, за счет этого увеличивается векторизуемость ценой ухудшения эффективности кэшей
- Отличие есть только в реализации SWAP фазы

PAM-1 и PAM-2 (2)

PAM-1	PAM-2
Объем вспомогательных данных	
$3n$	nk
Логическое условие при вычислении C_{jih}	

`if (val1 > val2){`
`  delta += val2 - val1;`
`} else if ( val3 == const1 ) {`
`  if ( val2 < val4 ){`
`  delta += val2 - val1;`
`  } else {`
`  delta += val4 - val1;`
`  }`
`}`
	`if (val1 > val2){`
	`  delta += val2 - val1;`
	`}`

Эксперименты

- Аппаратная платформа
  - Intel Xeon Phi 60 cores
  - Intel Xeon 12 cores
- Параметры экспериментов
  - Тип данных: вещественные одинарной точности
  - Ограничение на количество объектов (ОЗУ Intel Xeon Phi): 40 тыс.
- Цели
  - Сравнить скорость работы алгоритма PAM на CPU и Intel Xeon Phi

Режим работы Intel Xeon Phi*

Multi-Core Hosted
Как последовательный, так и параллельный код

Symmetric
Сбалансированные потребности исполнения

Offload
Последовательный код с высоко параллельными блоками

Many-Core Hosted
Высоко параллельный код

- В исследовании используется режим offload, так как в будущем предполагается вызов кода из программ исполняемых на CPU

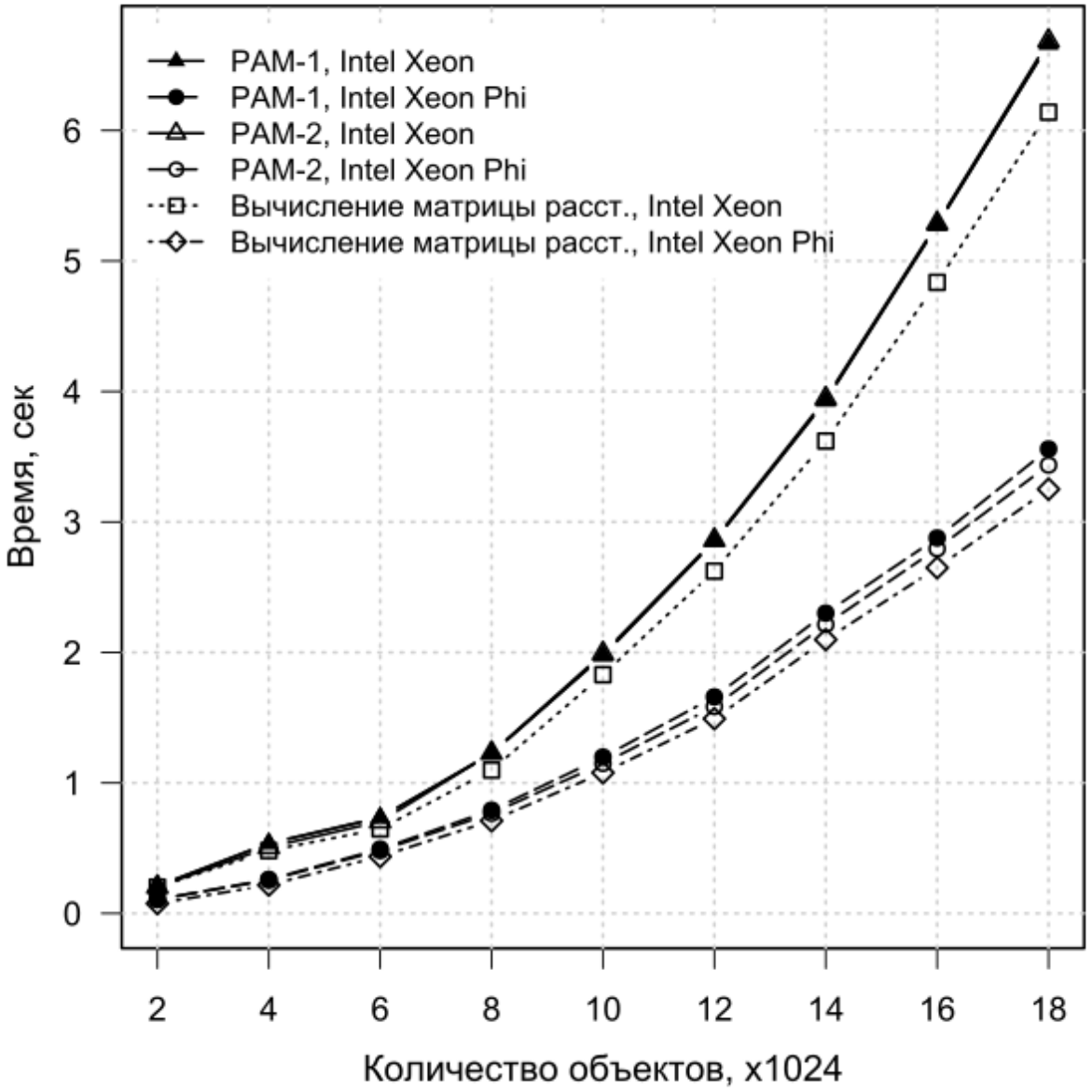
*по материалам Colfax Research

Характеристики данных

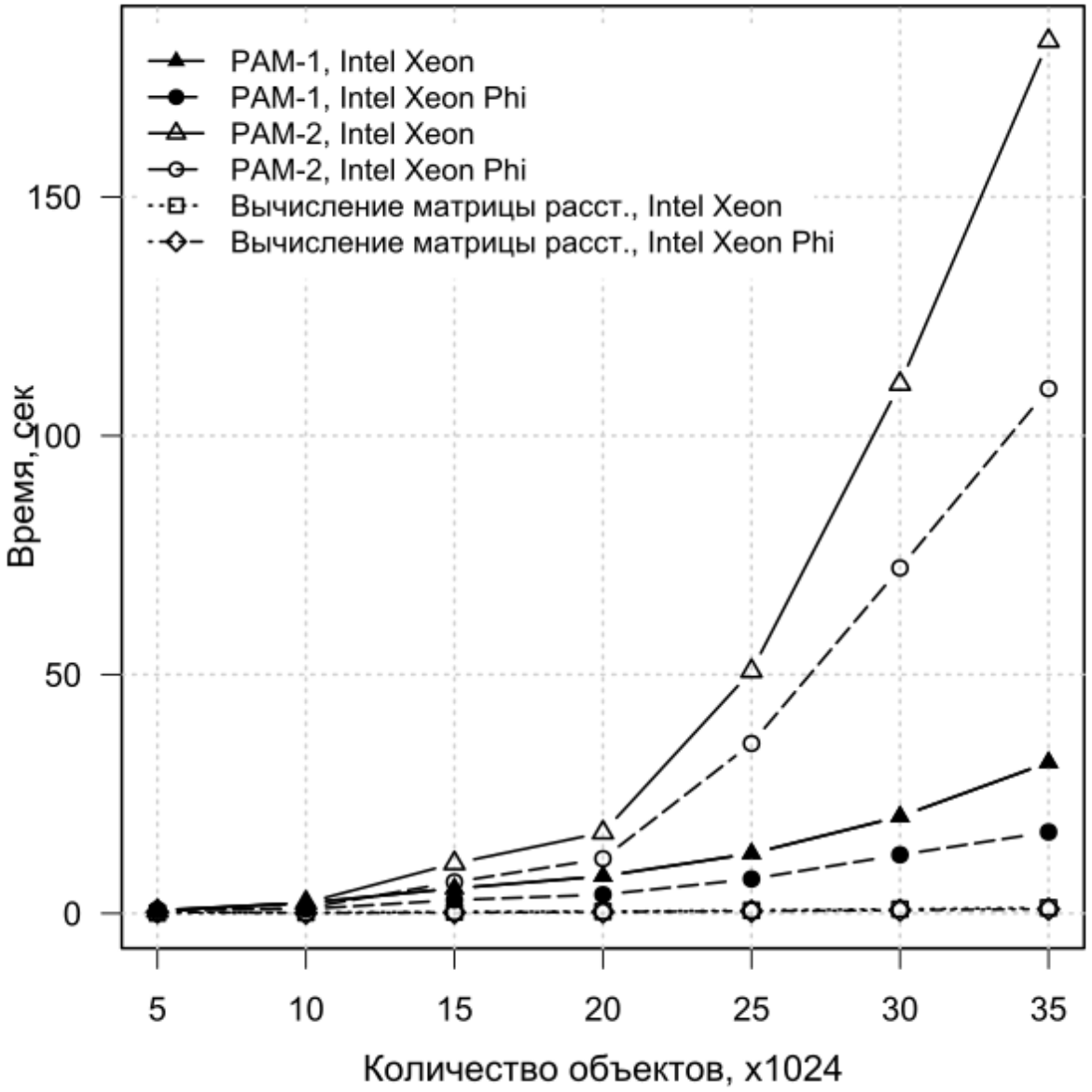
Набор данных	p	k	n, $\times 2^{10}$	
			min	max
FCS Human	423	10	2	18
Corel Image Histogram	32	10	5	35
MixSim	5	10	5	35
Letters	16	26	2	18

- p – размерность пространства
- k – количество кластеров, которые искал PAM
- n – количество кластеризуемых объектов

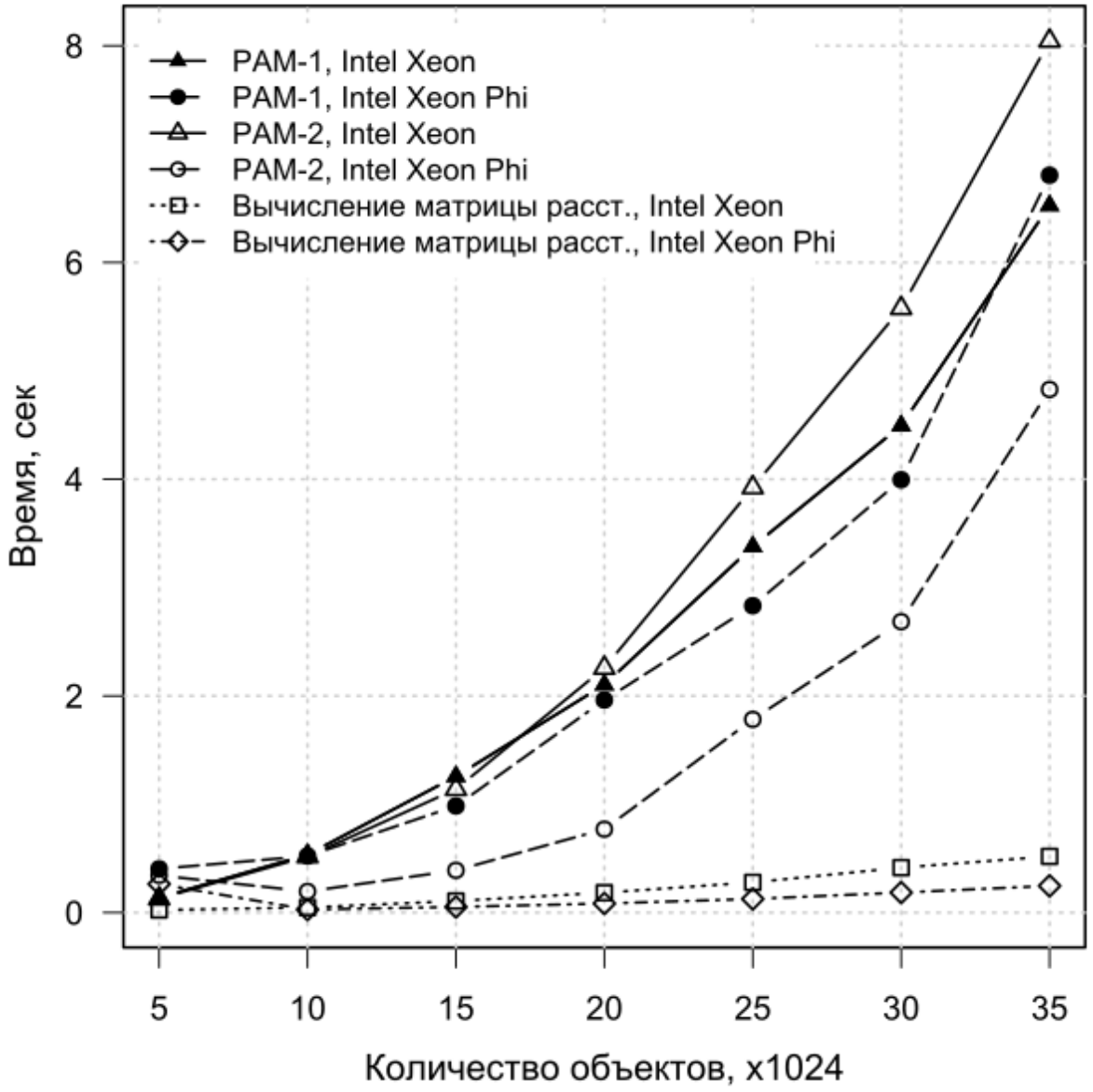
Набор данных FCS Human



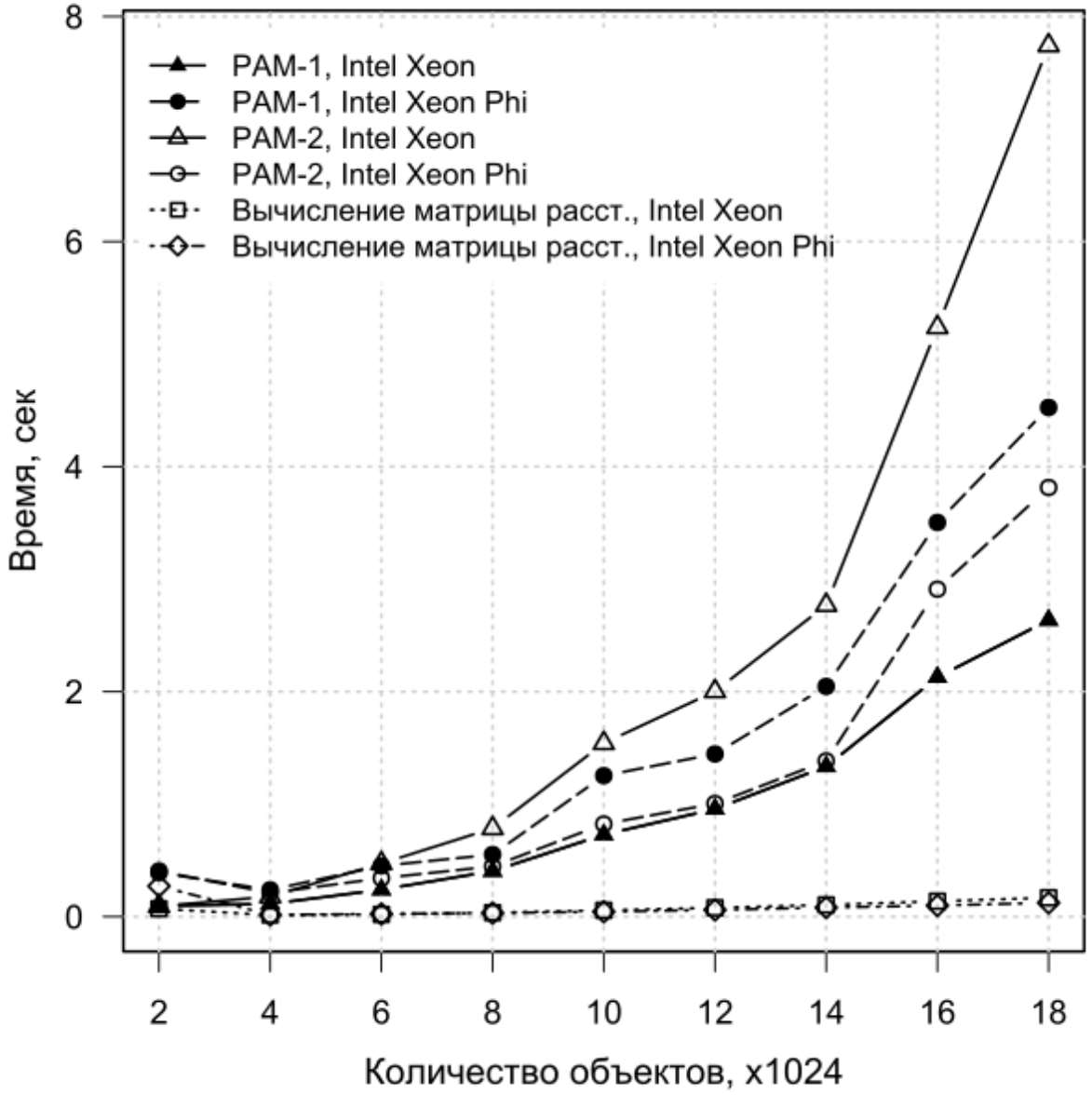
Набор данных гистограммы Corel



Набор данных MixSim



Набор данных Letter



CAMPAIGN

- Kohlhoff K. J. et al. CAMPAIGN: an open-source library of GPU-accelerated data clustering algorithms //Bioinformatics. – 2011. – Т. 27. – №. 16. – С. 2321-2322.
- URL: https://simtk.org/home/campaign
- «Целью проекта CAMPAIGN является модуляризация и параллелизация алгоритмов кластеризации и новые подходы к кластеризации, с упором на исполнение на GPU.»
- Последняя версия: 30 января 2011 г.

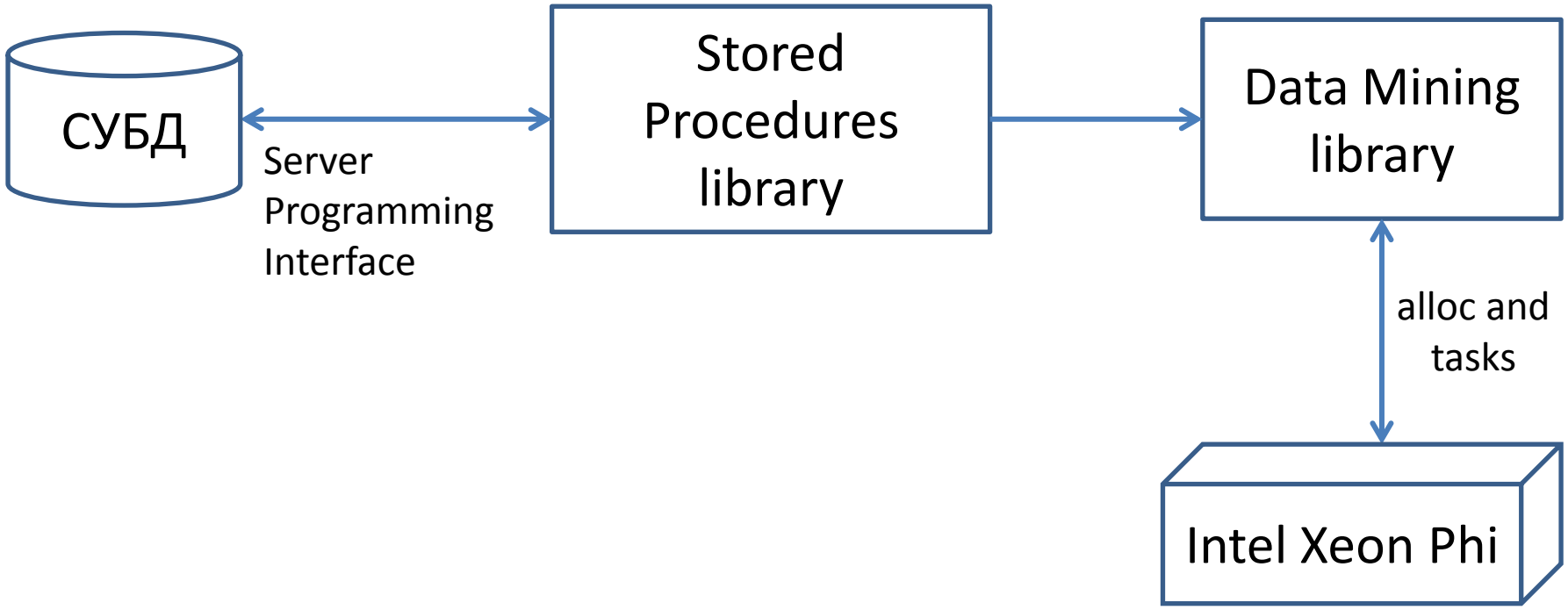
CAMPAIGN: PAM для GPU

	PAM	CAMPAIGN
Вычисление матрицы расстояний	Используется	Не используется
Инициализация медоидов	BUILD фаза – выбор минимального распределения	Первые *k* кластеризуемых объектов
Перемещение медоидов	SWAP фаза – поиск лучшего перемещения	Поиск первого перемещения, улучшающего *E*
Критерий останова	Останов, когда E нельзя более улучшить	Останов по прошествии фиксированного числа итераций

Итоги

- В работе описана параллельная версия алгоритма кластеризации Partitioning Around Medoids для сопроцессора Intel Xeon Phi
  - OpenMP
  - Векторизация
  - Тайлинг
- Представлены результаты вычислительных экспериментов, подтверждающих эффективность разработанного алгоритма
- Эксперименты показали, что скорость работы алгоритма PAM определяется природой кластеризуемых данных

Продолжение исследования



Продолжение исследования (2)