Исследование производительности реализации метода lADE в системе фрагментированного программирования LuNA

Киреев С.Е., Перепёлкин В.А.
ИВМиМГ СО РАН

Введение

- Технология фрагментированного программирования (ТФП)
  - Нацелена на автоматизацию решения больших численных задач на суперкомпьютерах
- Система фрагментированного программирования LuNA
  - реализует ТФП

Цель работы

- Проанализировать производительность системы LuNA на модельной задаче решения уравнения теплопроводности модификацией метода IADE*

*Norma Alias, M.S. Sahimi, A.R. Abdullah «Parallel strategies for the iterative alternating decomposition explicit interpolation-conjugate gradient method in solving heat conductor equation on a distributed parallel computer systems» // Proceedings of The 3rd International Conference On Numerical Analysis in Engineering, Vol. 3, 2003, p. 31-38.

Модельная задача

- Метод IADE (Iterative Alternating Decomposition Explicit)
  - Используется для решения параболических уравнений
  - На каждом шаге по времени представляет собой двухстадийный итерационный процесс:

$$u_i^{(p+1/2)} = (-l_{i-1}u_{i-1}^{(p+1/2)} + s_i u_i^{(p)} + w_i u_{i+1}^{(p)} + f_i)/d$$

$$u_{m+1-i}^{(p+1)} = (v_{m-1}u_{m-i}^{(p+1/2)} + su_{m+1-i}^{(p+1/2)} + gf_{m+1-i} - u_{m+1-i}u_{m+2-i}^{(p+1)})/d_{m+1-i}$$

- Для получения параллельного алгоритма используется модификация метода – красно-черное упорядочение элементов (RB)
- Дополнительно используется способ ускорения сходимости итерационного процесса на основе метода сопряженных градиентов (CG)

Модельная задача

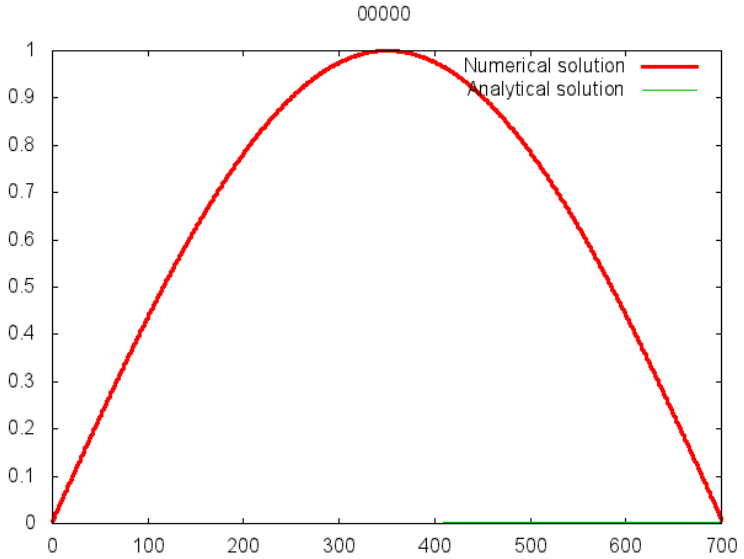
- Решается одномерное уравнение теплопроводности

$$\frac{\partial U}{\partial t} = \frac{\partial^2 U}{\partial x^2} \quad 0 \le x \le 1$$

граничные условия: $U(0, t) = U(1, t) = 0$

начальное условие: $U(x, 0) = \sin(\pi x) \quad 0 \le x \le 1$

- Решение:



Алгоритм (1): IADE

```
for(it = 0; it < NT; it++) {
  for(j=1;j<=N;j++) f[j] = λ/(1-θ)*(x[j-1]+x[j+1]) + (1 - 2λ(1 - θ))*x[j];

  Δ = ε;
  while(Δ >= ε) {

    for(j=1;j<=N;j++)
      y[j] = y[j] + ω_y(-d*y[j]-lo[j-1]*y[j-1]+s[j]*x[j]+w*x[j+1]+f[j])/ d ;


    for(j=1;j<=N;j++)
      x[j] = x[j] + ω_z(-d[j]*x[j]+v[j-1]*y[j-1]+s*y[j]+g*f[j]-u*x[j+1])/d[j];









    Δ = max( max_{j=1..N} |xp[j]-x[j]| );
    for(j=1;j<=N;j++) xp[j] = x[j];
  }

}
```

Алгоритм (2): IADE+RB

```
for(it = 0; it < NT; it++) {
  for(j=2;j<=N;j+=2) f[j] = λ/(1-θ)*(x[j-1]+x[j+1]) + (1 - 2λ(1 - θ))*x[j];
  for(j=1;j<=N;j+=2) f[j] = λ/(1-θ)*(x[j-1]+x[j+1]) + (1 - 2λ(1 - θ))*x[j];
  Δ = ε;
  while(Δ >= ε) {
    for(j=2;j<=N;j+=2)
      y[j] = y[j] + ω_y (- d *y[j]-lo[j-1]*y[j-1]+s[j]*x[j]+ w *x[j+1]+f[j]) / d;
    for(j=2;j<=N;j+=2)
      x[j] = x[j] + ω_z (-d[j]*x[j]+(v[j-1]*y[j-1]+ s *y[j]+ g *f[j])- u *x[j+1])/d[j];

    for(j=1;j<=N;j+=2)
      y[j] = y[j] + ω_y (- d *y[j]-lo[j-1]*y[j-1]+s[j]*x[j]+ w *x[j+1]+f[j]) / d;
    for(j=1;j<=N;j+=2)
      x[j] = x[j] + ω_z (-d[j]*x[j]+v[j-1]*y[j-1]+ s *y[j]+ g *f[j]- u *x[j+1])/d[j];



    Δ = max( max_{j=1..N} |xp[j]-x[j]| );
    for(j=1;j<=N;j++) xp[j] = x[j];
  }

}
```

Алгоритм (3): IADE+RB+CG

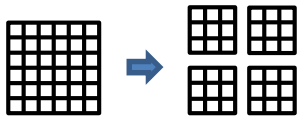
```
for(it = 0; it < NT; it++) {
  for(j=2;j<=N;j+=2) f[j] = λ/(1−θ)*(x[j-1]+x[j+1]) + (1 − 2λ(1 − θ))*x[j];
  for(j=1;j<=N;j+=2) f[j] = λ/(1−θ)*(x[j-1]+x[j+1]) + (1 − 2λ(1 − θ))*x[j];
  Δ = ε;
  while(Δ >= ε) {
    for(j=2;j<=N;j+=2)
      y[j] = y[j] + ω_y (- d *y[j]-lo[j-1]*y[j-1]+s[j]*x[j]+ w *x[j+1]+f[j]) / d;
    for(j=2;j<=N;j+=2)
      x[j] = x[j] + ω_z (-d[j]*x[j]+(v[j-1]*y[j-1]+ s *y[j]+ g *f[j])- u *x[j+1])/d[j];

    for(j=1;j<=N;j+=2)
      y[j] = y[j] + ω_y (- d *y[j]-lo[j-1]*y[j-1]+s[j]*x[j]+ w *x[j+1]+f[j]) / d;
    for(j=1;j<=N;j+=2)
      x[j] = x[j] + ω_z (-d[j]*x[j]+v[j-1]*y[j-1]+ s *y[j]+ g *f[j]- u *x[j+1])/d[j];

    for(j=2;j<=N;j+=2) x[j],r[j],z[j] = cgR(x[j], r[j], z[j-1], z[j], z[j+1]);
    for(j=1;j<=N;j+=2) x[j],r[j],z[j] = cgB(x[j], r[j], z[j-1], z[j], z[j+1]);

    Δ = max( max_{j=1..N} |xp[j]-x[j]| );
    for(j=1;j<=N;j++) xp[j] = x[j];
  }

}
```

Алгоритм (4): с декомпозицией

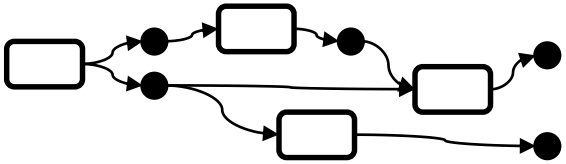
```
for(it = 0; it < NT; it++) {
  for(j=2;j<=N;j+=2) f[j] = λ/(1−θ)*(x[j-1]+x[j+1]) + (1 − 2λ(1 − θ))*x[j];
  for(j=1;j<=N;j+=2) f[j] = λ/(1−θ)*(x[j-1]+x[j+1]) + (1 − 2λ(1 − θ))*x[j];
  Δ = ε;
  while(Δ >= ε) {
    for(j=2;j<=N;j+=2)
      y[j] = y[j] + ω_y (- d *y[j]-lo[j-1]*y[j-1]+s[j]*x[j]+ w *x[j+1]+f[j]) / d;
    for(j=2;j<=N;j+=2)
      x[j] = x[j] + ω_z (-d[j]*x[j]+(v[j-1]*y[j-1]+ s *y[j]+ g *f[j])- u *x[j+1])/d[j];
    send_border_right(y);
    for(j=1;j<=N;j+=2)
      y[j] = y[j] + ω_y (- d *y[j]-lo[j-1]*y[j-1]+s[j]*x[j]+ w *x[j+1]+f[j]) / d;
    for(j=1;j<=N;j+=2)
      x[j] = x[j] + ω_z (-d[j]*x[j]+v[j-1]*y[j-1]+ s *y[j]+ g *f[j]- u *x[j+1])/d[j];
    send_border_left(z); send_border_right(z);
    for(j=2;j<=N;j+=2) x[j],r[j],z[j] = cgR(x[j], r[j], z[j-1], z[j], z[j+1]);
    for(j=1;j<=N;j+=2) x[j],r[j],z[j] = cgB(x[j], r[j], z[j-1], z[j], z[j+1]);
    send_border_left(x);
    Δ = max( max_{j=1..N} |xp[j]-x[j]| ); Allreduce(Δ);
    for(j=1;j<=N;j++) xp[j] = x[j];
  }
  send_border_right(xR);
}
```

Реализация задачи в системе LuNA

- Этапы реализации:
  1. Фрагментировать данные и вычисления



  2. Представить фрагментированный алгоритм в виде графа зависимостей

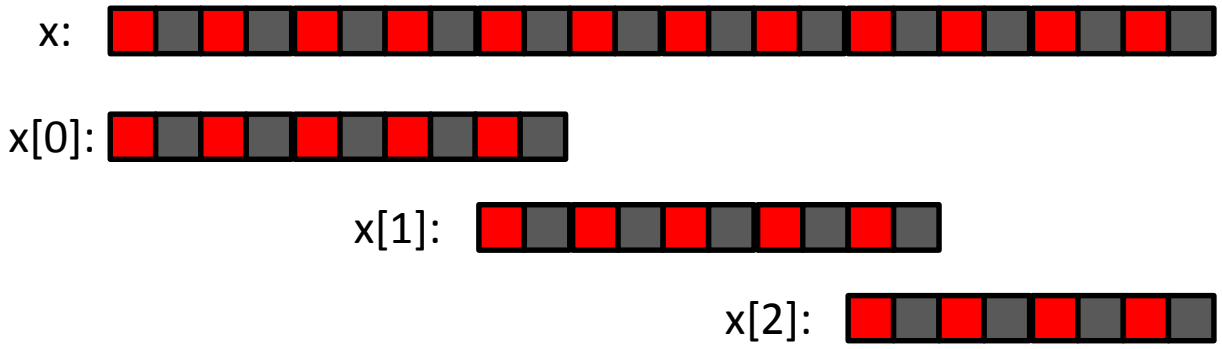


  3. Записать алгоритм на языке LuNA

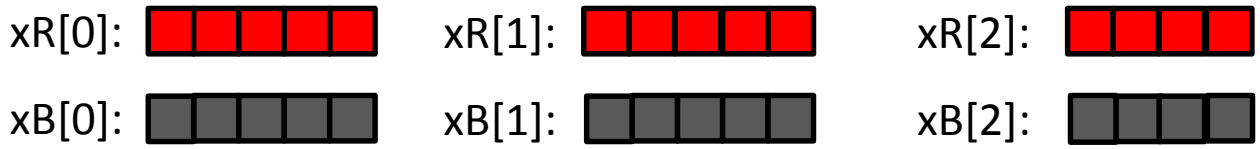
  4. Спланировать размещение фрагментов данных, место и порядок выполнения фрагментов вычислений

Реализация задачи в системе LuNA

- Фрагментация алгоритма
  - массивы разделены на фрагменты одинакового размера (с перекрытием),



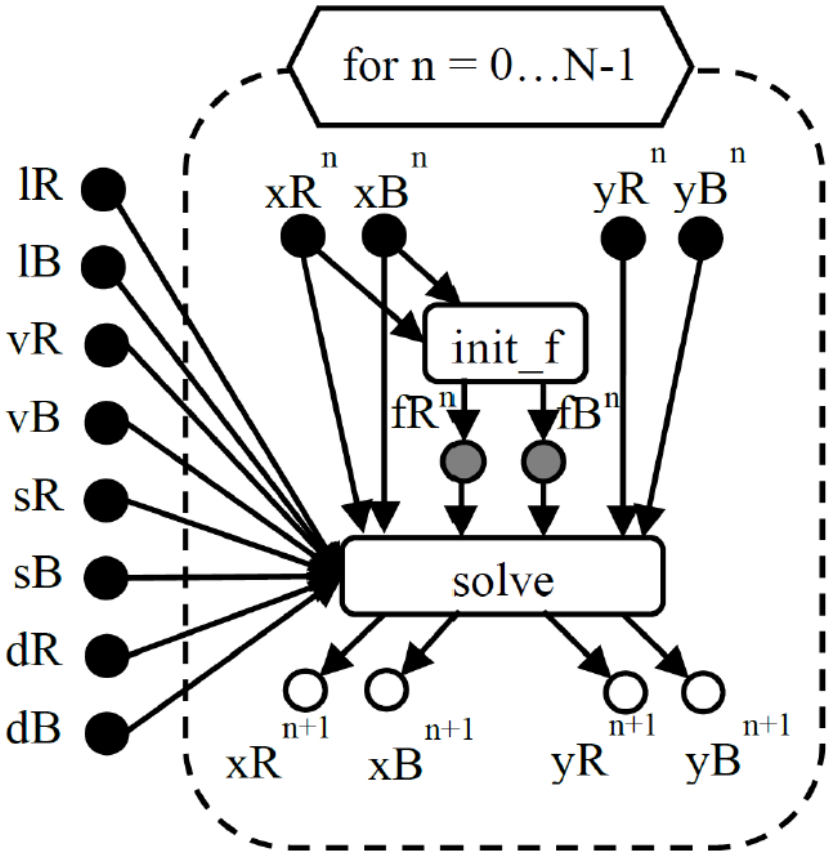
  - красные (Red) и черные (Black) элементы массивов сгруппированы в отдельные фрагменты,



  - таким же образом сгруппированы и операции.

Реализация задачи в системе LuNA

- Схема алгоритма (1) – шаги по времени



- фрагменты данных (переменные единственного присваивания), здесь это фрагментированные массивы

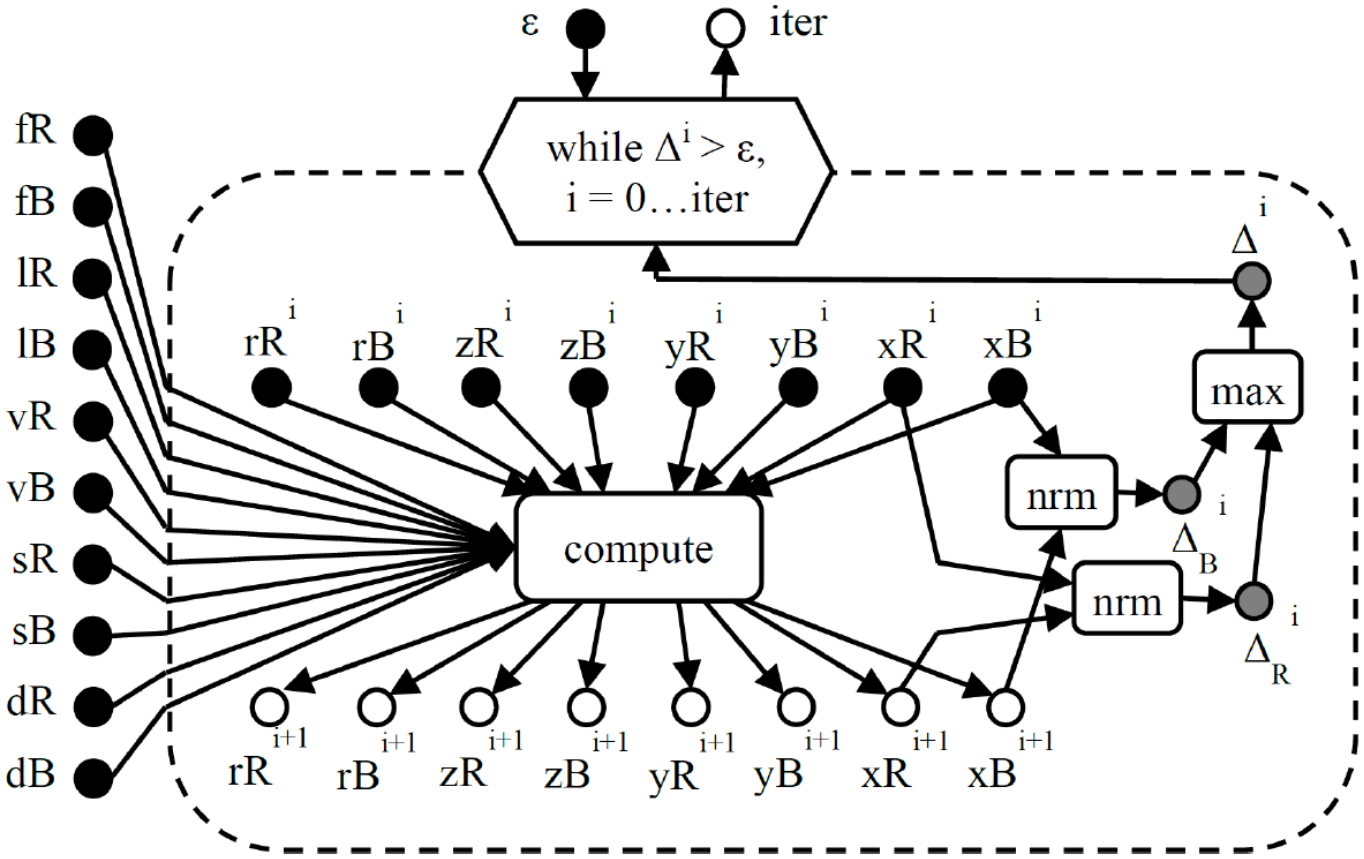
- фрагменты вычислений (операции однократного срабатывания), здесь это массовые операции

- информационные зависимости (отношения вход-выход)

- оператор «индексированное множество»

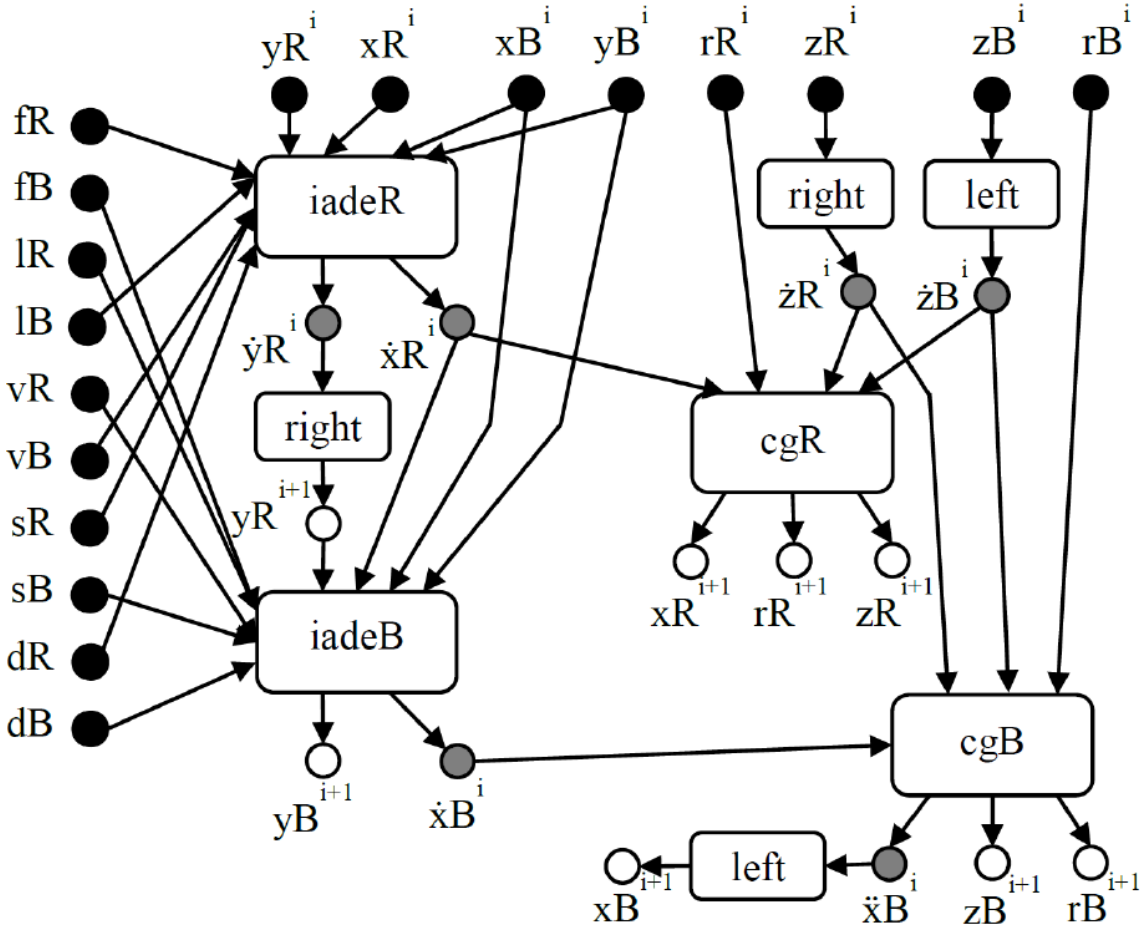
Реализация задачи в системе LuNA

- Схема алгоритма (2) – операция «solve»

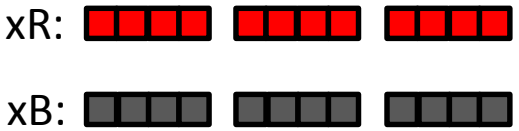


Реализация задачи в системе LuNA

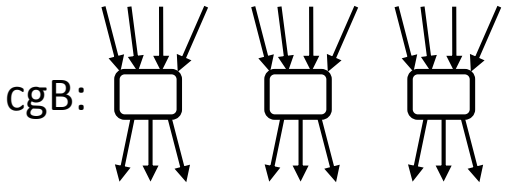
- Схема алгоритма (3) – операция «compute»



Все переменные-массивы здесь фрагментированы:



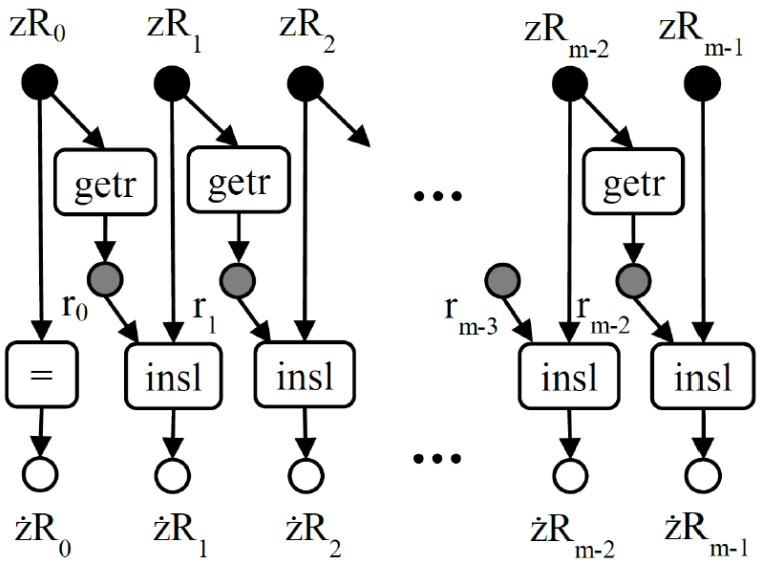
Фрагменты операций iadeR, iadeB, cgR, cgB работают с фрагментами массивов только со «своими» индексами:



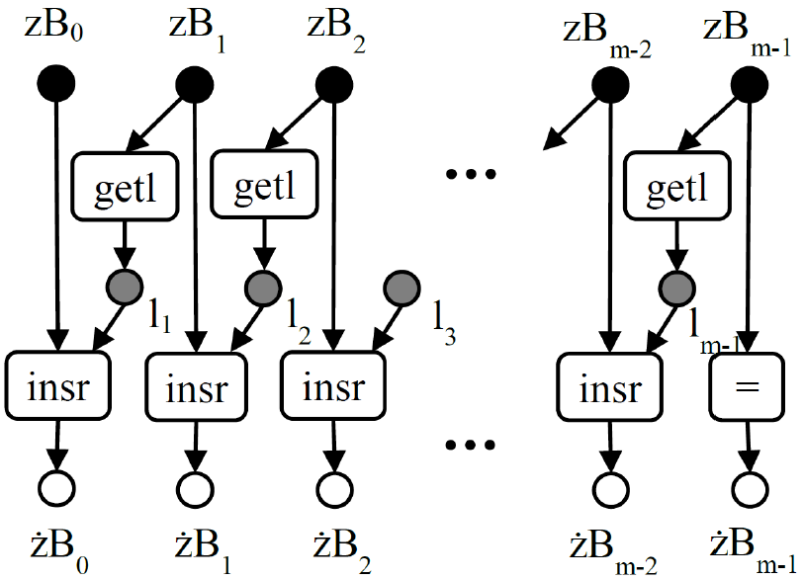
Реализация задачи в системе LuNA

- Схема алгоритма (4) – обмен границами
  - Взаимодействие между фрагментами с разными индексами происходит только в операциях «left» и «right»:

Операция «right»

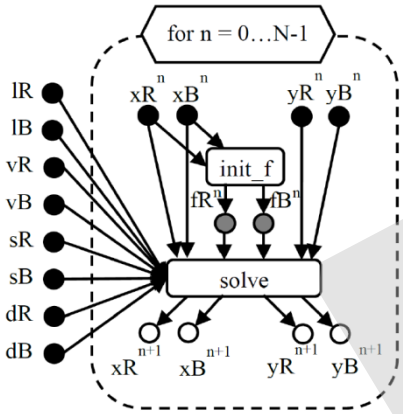


Операция «left»

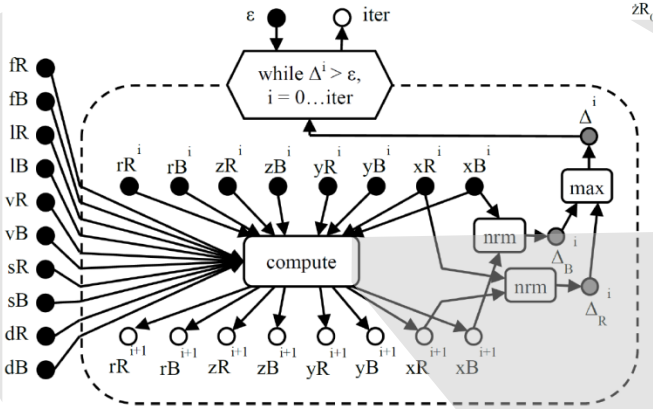


Реализация задачи в системе LuNA

- Схема алгоритма

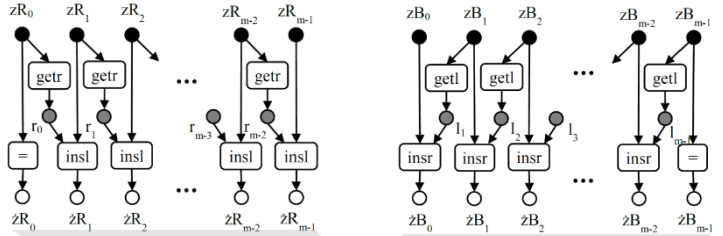


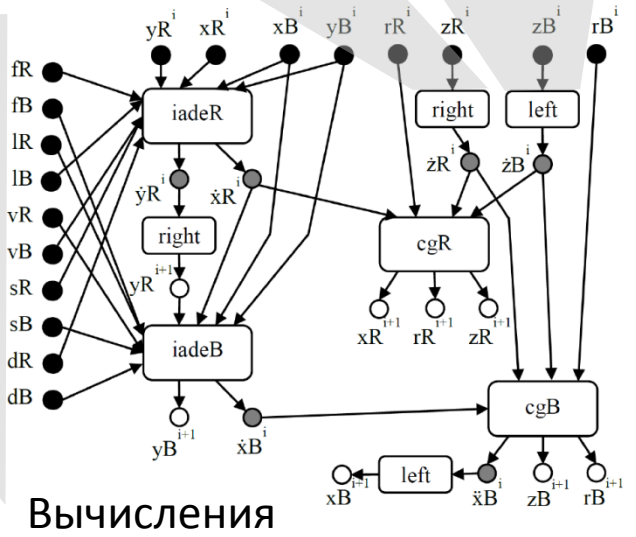
Шаги по времени



Итерации до сходимости

Обмен границами





Вычисления

Реализация задачи в системе LuNA

- Запись программы на языке LuNA: 2 части

Фрагментированный алгоритм
на языке LuNA

```
sub send_all_R_right_borders(#in int nf, name nfsR, name xR_in,
                             #out name xR_out)
{ df xright;
  for j=0..nf-2
    cf get[j]: get_R_right_border(#in nfsR[j],xR_in[j],
                                  #out xright[j]);
  cf cp[0]: copy(#in xR_in[0], #out xR_out[0]);
  for j=1..nf-1
    cf set[j]: set_R_left_border(#in nfsR[j],xR_in[j],xright[j-1],
                                 #out xR_out[j]) --> (xright[j-1]);
}

sub send_all_B_left_borders(#in int nf, name nfsB, name xB_in,
                            #out name xB_out)
{ df xleft;
  for j=1..nf-1
    cf get[j]: get_B_left_border(#in nfsB[j],xB_in[j],
                                 #out xleft[j]);
  cf cp[nf-1]: copy(#in xB_in[nf-1], #out xB_out[nf-1]);
  for j=0..nf-2
    cf set[j]: set_B_right_border(#in nfsB[j],xB_in[j],xleft[j+1],
                                  #out xB_out[j]) --> (xleft[j+1]);
}
```

Текстовое представление графа

Фрагменты кода (чистые функции)
на языке C++

```
void c_get_B_left_border(int nfsB,const InputDF& df_xB,
                                  OutputDF& df_xleft)
{ assert(df_xB.getSize() == (nfsB+1)*sizeof(double));
  const double *xB = df_xB.getData<double>();
  df_xleft.setValue(xB[0]);
}
void c_get_R_right_border(int nfsR,const InputDF& df_xR,
                                   OutputDF& df_xright)
{ assert(df_xR.getSize() == (nfsR+1)*sizeof(double));
  const double *xR = df_xR.getData<double>();
  df_xright.setValue(xR[nfsR]);
}
void c_set_B_right_border(int nfsB,const InputDF& df_xB,
                          double xleft,OutputDF& df_xB_new)
{ assert(df_xB.getSize() == (nfsB+1)*sizeof(double));
  const double *xB = df_xB.getData<double>();
  double *xnB = df_xB_new.create<double>(nfsB+1);
  for (int i=0;i<=nfsB-1;i++) xnB[i] = xB[i];
  xnB[nfsB] = xleft;
}
void c_set_R_left_border(int nfsR,const InputDF& df_xR,
                         double xright,OutputDF& df_xR_new)
{ assert(df_xR.getSize() == (nfsR+1)*sizeof(double));
  const double *xR = df_xR.getData<double>();
  double *xnR = df_xR_new.create<double>(nfsR+1);
  xnR[0] = xright;
  for (int i=1;i<=nfsR;i++) xnR[i] = xR[i];
}
```

Реализация системы фрагментированного программирования LuNA

- Система LuNA. Полуавтоматическое распределение ресурсов и управление:
  - Задание отображения ФД и ФВ на узлы с помощью модуля Path-Finder.
  - Автоматическая пересылка ФД на нужный узел.
  - Автоматическое исполнение фрагментов вычислений в несколько потоков с учетом зависимостей по данным.

Реализация системы фрагментированного программирования LuNA

- Система LuNA-fw
  - Использует фрагменты кода LuNA-программы
  - Явно заданная управляющая программа в событийной модели

События	Действия
• Старт программы • Завершение выполнения ФВ • Вычисление ФД • Получение ФД по сети	• Запустить ФВ • Переслать ФВ на заданный узел • Уничтожить ФД

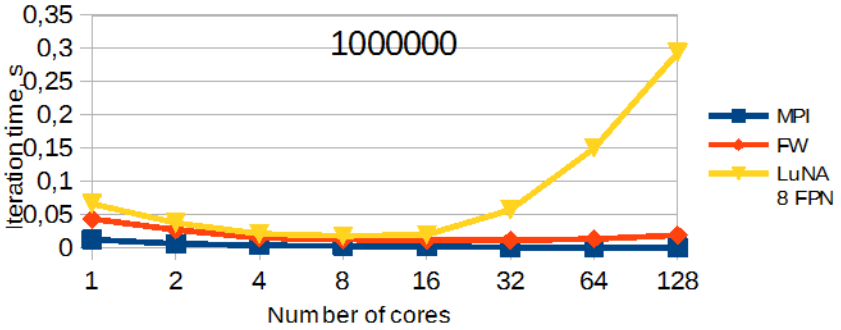
Реализация системы фрагментированного программирования LuNA

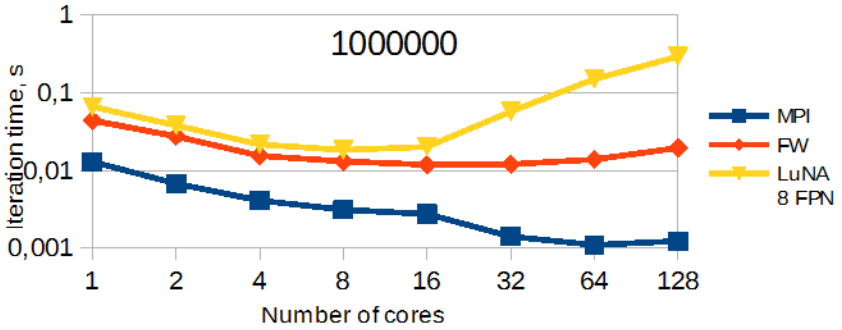
- Система LuNA-fw – явное задание управляющей программы

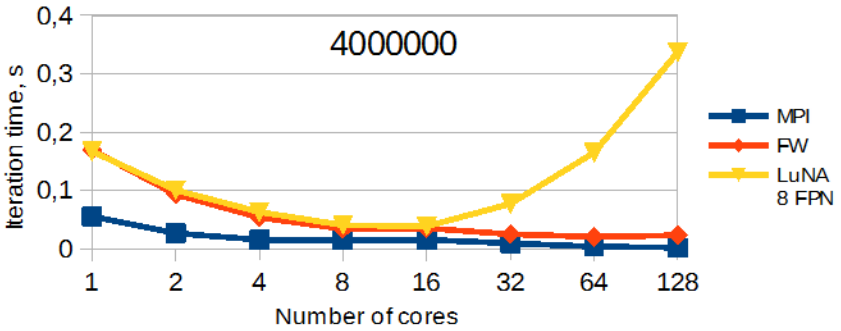
Реализация задачи в системе LuNA

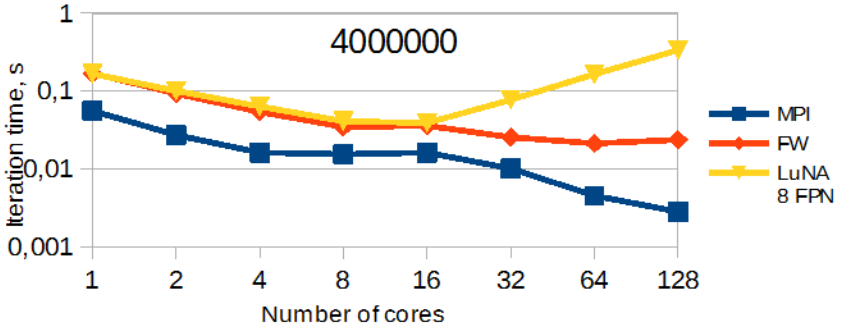
	LuNA	LuNA-fw	MPI
Распределение ресурсов	Выполняется один процесс на узел. Фрагменты каждого фрагментированного массива равномерно распределяются по процессам с учетом соседства.	Выполняется один процесс на ядро. На каждый процесс приходится один фрагмент массива.	Выполняется один процесс на ядро. На каждый процесс приходится один фрагмент массива.
Управление порядком исполнения	Фрагменты вычислений каждого процесса выполняются параллельно несколькими рабочими потоками по мере готовности данных.	Фрагменты вычислений в каждом процессе выполняются последовательно в порядке, заданном управляющей программой.	В каждом процессе последовательное выполнение с фиксированным порядком.

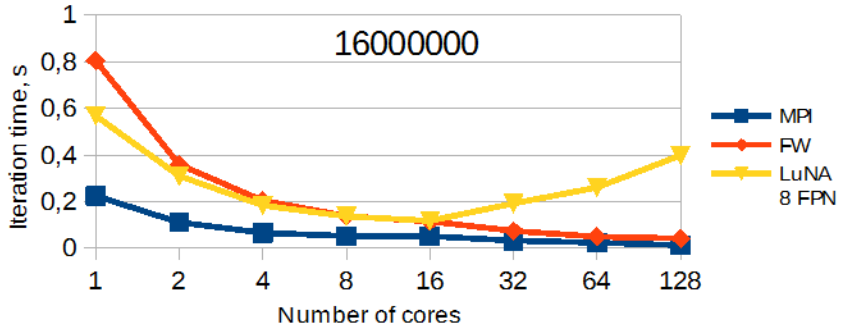
Сравнение реализаций

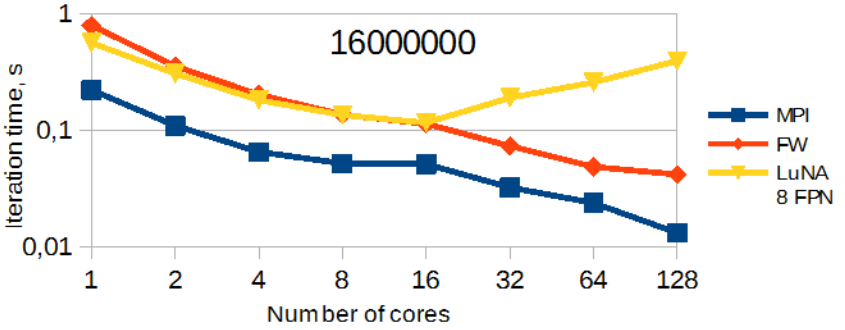






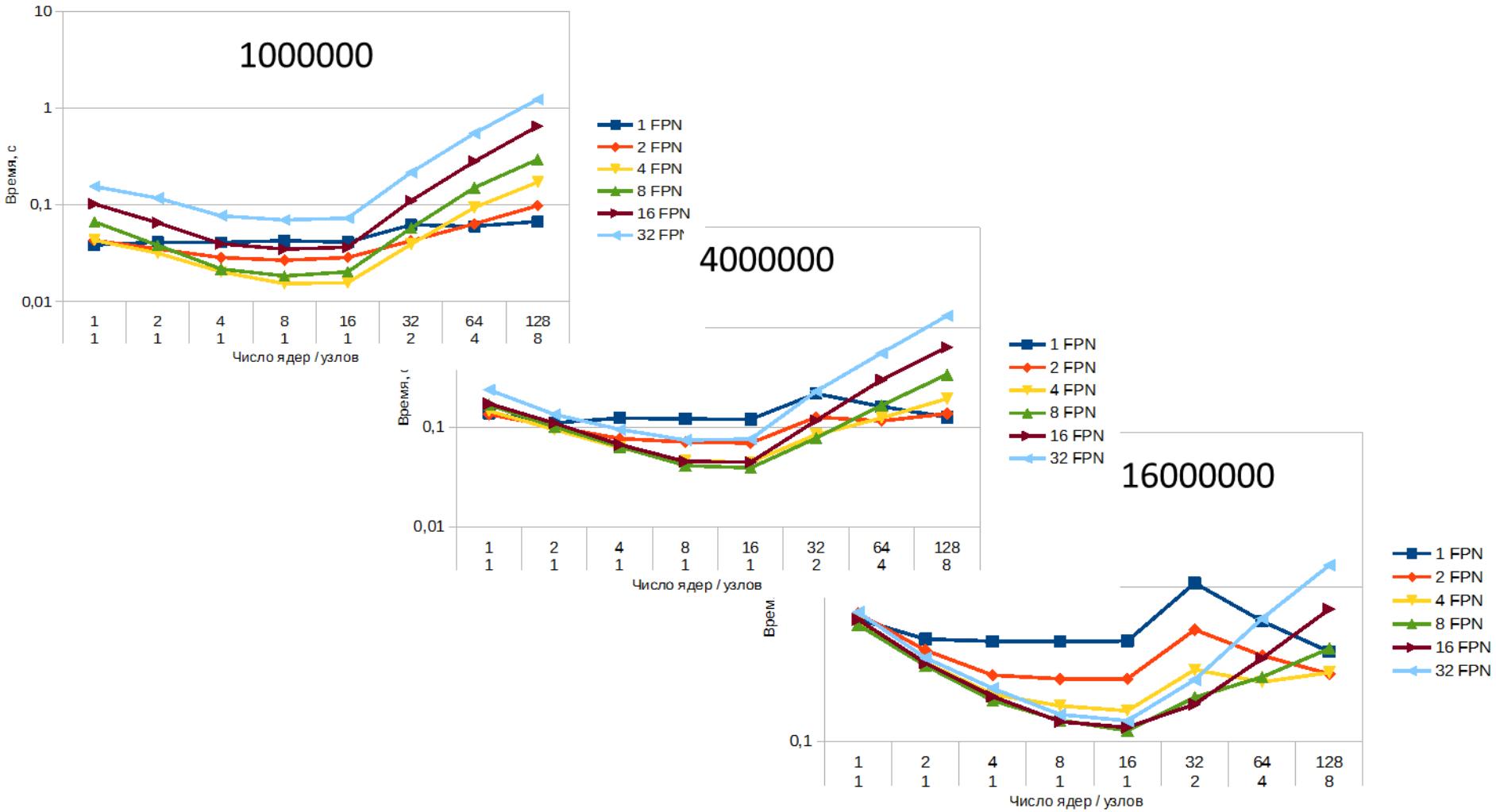




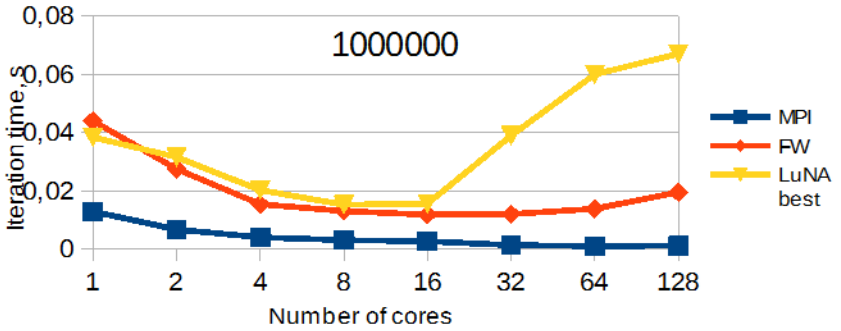


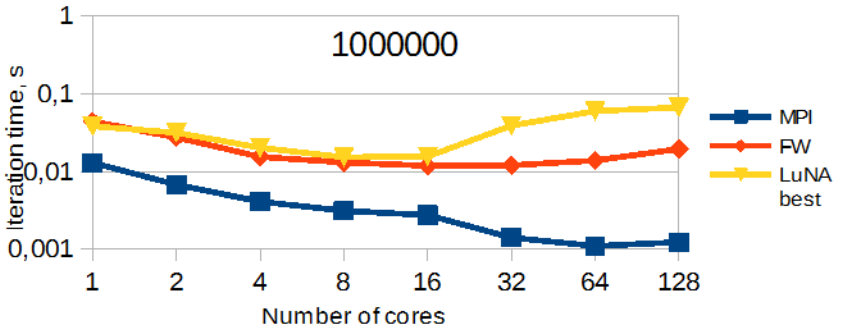
Оценка производительности

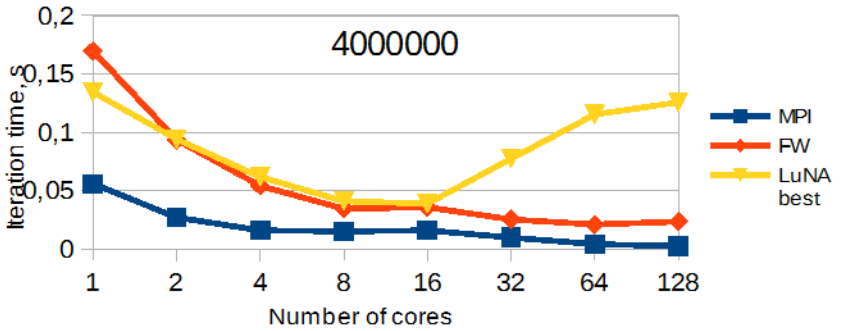
- LuNA: влияние степени фрагментации

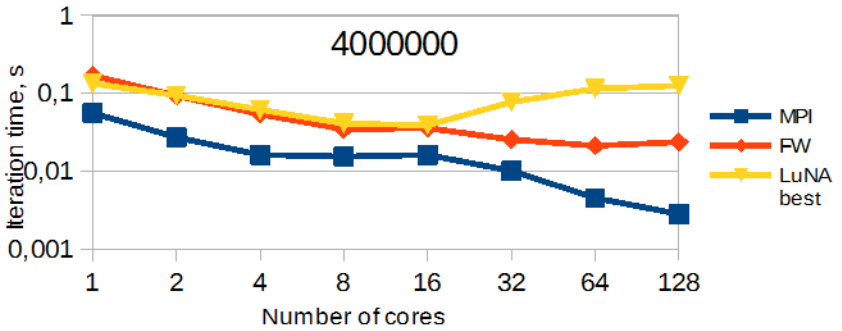


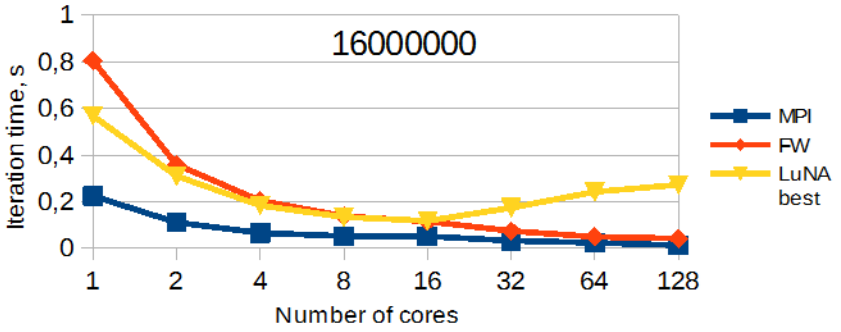
Сравнение реализаций

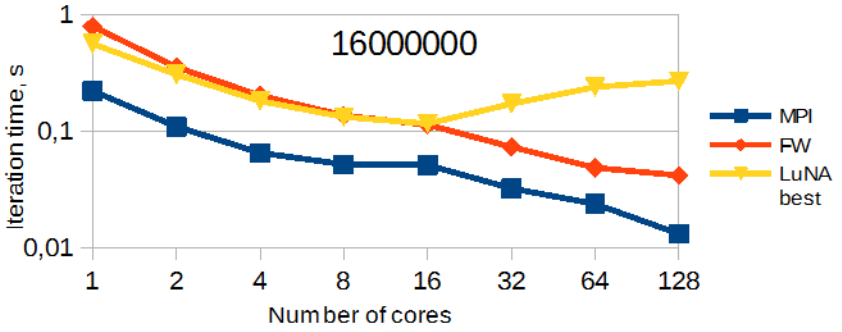












Заключение

- Система LuNA достигает приемлемой эффективности (сравнимой с LuNA-fw) в при работе в пределах одного узла.
- Степень фрагментации влияет на производительность. При чрезмерной фрагментации производительности системы LuNA сильно деградирует.

Спасибо за внимание!