УДК 004.94+547.022

РЕАЛИЗАЦИЯ МУРАВЬИНОГО АЛГОРИТМА ДЛЯ ИЗУЧЕНИЯ ФОЛДИНГА БЕЛКОВ НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ

Базаров Р.К.
младший научный сотрудник Центра разработки программных продуктов и аппаратно-программных комплексов при Ташкентском университете информационных технологий,
тел.: +(99897) 187-48-05, e-mail: rustam.bazarov@gmail.com

Задача поиска третичной структуры белка по его первичной аминокислотной последовательности (protein folding problem, PFP) – важнейшая задача структурной биологии. К сожалению, даже такая грубая модель, как HP-PFP-2, учитывающая только гидрофобные взаимодействия аминокислотных остатков на двумерной решетке, является NP-сложной и может успешно решаться только эвристическими методами глобальной оптимизации, например, муравьиным алгоритмом. В статье исследуются способы модификации и распараллеливания муравьиного алгоритма для задачи фолдинга белков. Подробно описана программная реализация параллельного муравьиного алгоритма на графических процессорах, обсуждаются результаты вычислительного эксперимента.

Ключевые слова: алгоритм оптимизации муравьиной колонией, фолдинг белков, графические процессоры.

THE IMPLEMENTATION OF ANT COLONY OPTIMIZATION ALGORITHM FOR THE STUDY OF PROTEIN FOLDING PROBLEM ON GRAPHICS PROCESSING UNITS

Bazarov R.K.

The task of finding the tertiary structure of the protein by its primary amino acid sequence (protein folding problem: PFP) – is a major challenge for structural biology. Unfortunately, even such a crude model as HP-PFP-2, which takes into account only hydrophobic interaction of amino acid residues on a two-dimensional lattice is NP-hard and can be successfully solved only by heuristic methods of global optimization, for example, the ant colony optimization algorithm. This article describres the methods of modification and parallelization of ant colony optimization algorithm for the protein folding problem. It is described in detail the software implementation of parallel ant colony optimization algorithm on the graphics processing units.

Keywords: the ant colony optimization algorithm, protein folding problem, graphics processing units.

OQSIL FOLDINGINI O'RGANISH UCHUN CHUMOLI ALGORITMINI GRAFIK PROTSESSORLARDA TATBIQ ETISH

Bazarov R.K.

Oqsilni uchlamchi tuzilishini uning daslabki aminokislotali ketma-ketligi (protein folding problem: PFP) bo'yicha izlash masalasi – strukturali biologiyaning eng muhim masalalaridan hisoblanadi. Afsuski, hattoki ikki o'lchovli to'rdagi aminokislota qoldiqlarini gidrofob ta`sirinigina hisobga oluvchi HP-PFP-2 kabi mukammal bo'lmagan modelning ham murakkablik darajasi NP hisoblanadi va faqatgina global optimallashtirishning evristik usullari bilan muvaffaqiyatli yechilishi mumkin, masalan, chumoli algoritmi yordamida. Maqolada oqsil foldingi masalasi uchun chumoli algoritmini takomillashtirish va parallellashtirish usullari o'rganilgan. Parallel chumoli algoritmini grafik protsessorlarga dasturiy tadbiqi to'liq tavsiflangan va hisoblash eksperimenti natijalari muhokama qilingan.

Tayanch iboralar: oqsil foldingi, chumolilar koloniyasini optimallashtirish algoritmi, grafik protsessorlar.

1. Введение

Задача предсказания структур белков (protein folding problem) является центральной задачей биоинформатики. Выборка аминокислот и их последовательность дают индивидуальные характеристики для каждого белка. Длина аминокислотной последовательности может варьироваться от 20 до 40000 аминокислотных остатков. Аминокислотная последовательность однозначно определяет уникальную трехмерную структуру белка. Определение третичной структуры белка по его первичной аминокислотной последовательности — важная фармакологическая задача, поскольку действие большинства лекарственных средств (низкомолекулярных веществ - лигандов) основано на молекулярном докинге (присоединении) к белковой макромолекуле. В результате докинга лиганд может либо стабилизировать нормально свернутую структуру последовательности, либо разрушить путь ее сворачивания, что приведет к построению вредоносных белков. Таким образом, знание точной

трехмерной структуры белка поможет в создании новых эффективных лекарств.

Сайрус Левинталь в 1968 г. оценил число возможных конформаций: 10^{100} для цепи из 100 аминокислотных остатков. Если переход из одной конформации в другую будет занимать 10^{-13} с, то полный перебор всех конформаций потребует 10^{80} лет. Этот парадокс Левинталь объяснил тем, что «самоорганизующийся белок следует по какому-то специальному пути сворачивания, и та структура, где этот путь заканчивается, и является его нативной структурой, не зависимо от того, есть ли еще более стабильная укладка цепи, или нет» [1].

Чтобы избежать проклятия размерности парадокса Левинталя, исследуют грубые модели: например HP-модель, согласно которой все аминокислоты делятся на полярные («P») и гидрофобные («H»). Водобоязнь неполярных молекул приводит к тому, что небольшие белковые цепи (около 150-200 остатков) формируют в водном окружении белковую глобулу с гидрофобным ядром. Этот физический принцип и использовал Кен Дилл для поиска наилучших (дающих минимум энергии) конформаций. Под энергией понимается число топологических контактов, образуемых между гидрофобными остатками в процессе сворачивания белка, взятое с отрицательным знаком. Конечная цель - найти полную свертку с минимумом энергии.

Таким образом, для заданной аминокислотной последовательности $s = s_1,s_2,...s_n$, требуется найти конформацию «*c*» с минимальной энергией:

$$c^* \in C(s) : E(c^*) = min\{E(c) | c \in C\},$$

где $C(s)$ - это набор всех возможных конформаций для s.

Степень свободы каждого остатка в цепи (число направлений сворачивания) ограничивают числом 3 - на плоскости и 5 - в пространстве. Это так называемые решетчатые модели (lattice bead models) [2, 3]. Но даже такие грубые модели могут быть затратными по времени. Например, для цепочки длиной $L = 71$ число конформаций на двумерной решетке составит $4.2x10^{30}$ [4]. Перебрать их не представляется возможным даже с использованием мощных вычислительных ресурсов.

Поэтому для решения таких задач используют алгоритмы роевого интеллекта, в частности алгоритмы муравьиных колоний [5].

Суть муравьиного алгоритма состоит в том, что при расширении конформации направление сворачивания выбирается с вероятностью

$$p_{ij}^k = \frac{\left[\tau_{ij}\right]^\alpha \cdot \left[\eta_{ij}\right]^\beta}{\sum_{j \in N_i^k} \left[\tau_{ij}\right]^\alpha \cdot \left[\eta_{ij}\right]^\beta},$$

где

$\tau_{i,j}$ = количество феромона на ребре (i, j);

$\eta_{i,j}$ = эвристическая информация ребра (i, j);

α, β = параметры, управляющие влиянием $\tau_{i,j}$ и $\eta_{i,j}$ соответственно;

j = достижимый узел графа из узла i, который еще не был посещен.

Феромоны - следы, оставляемые муравьями после нахождения оптимального решения, увеличивают вероятность расширения последовательности на один шаг в заданном направлении.

Эвристическая информация также увеличивает вероятность выбора направления с максимальным числом гидрофобных контактов. Она рассчитывается, как $\eta_{ij}=h_{ij}+1$, где h_{ij} - число H-H соседних позиций, которые возникнут в результате установки следующей аминокислоты. Следует отметить, что число образуемых гидрофобных контактов заранее неизвестно и определяется в процессе построения свертки.

Модификации муравьиного алгоритма рассмотрены в работах [6, 7], а вопросы сходимости - в [8, 9].

В [10] предлагается вариант решения той же задачи на треугольной решетке, в которой аминокислоты лучше упакованы, чем в квадратной. В той же статье предлагается способ обновления матрицы феромонов, учитывающий взаимодействия между удаленными остатками, благодаря которому алгоритм в среднем находит решение в 1,5 раза лучше. Однако, данная решетка менее исследована, что ограничивает возможность экспериментального сравнения приводимого в статье алгоритма с известными.

Сократить время получения суботпимального решения можно с помощью параллельной программной реализации алгоритма [11, 12].

В [13] описан AntMinerGPU алгоритм для решения задачи классификации, реализованный на графических процессорах.

Распараллеливание муравьиного алгоритма можно организовать по колониям муравьев [14,15], по отдельным муравьям (агентский подход) [16], по операциям, выполняемым каждым муравьем [17]. В данной статье описан последний способ. Сравнение всех перечисленных выше подходов приведено в [18].

Итак, пусть имеется G независимых колоний с M муравьями в каждой. Решение задачи состоит из следующих этапов: генерация точек инициации сворачивания, инициализация таблицы феромонов, вычисление эвристической информации, вычисление вероятности выбора направлений сворачивания белка, увеличение частичной свертки на одну позицию, вычисление энергии частичной свертки на данном шаге, обновление таблицы феромонов после построения тура, инициализация муравьев для следующего поколения. Суть предлагаемого способа распараллеливания рассматриваемого алгоритма состоит в том, чтобы муравьи одной колонии выполняли все эти этапы одновременно.

Для этой цели целесообразно использовать графические процессоры.

2. Материалы и методы

Графические процессоры (Graphics Processing Unit, GPU) представляют собой универсальные вычислительные модули, обеспечивающие многопотоковое параллельное программирование.

Чип GPU содержит несколько графических кластеров (graphic clusters) из вычислительных блоков - потоковых мультипроцессоров (stream multiprocessors), каждый из которых, в свою очередь, состоит из набора скалярных процессоров (scalar processors). Мультипроцессоры обмениваются информацией посредством медленной, глобальной памяти (global memory). Взаимодействие скалярных процессоров происходит благодаря быстрой памяти (shared memory), общей для процессоров одного мультипроцессора. Универсальные вычисления на GPUs компании NVIDIA обеспечивает технология программирования CUDA (Compute Unified Device Architecture) [19].

Параллельные части программы выполняются в виде так называемых ядерных функций (kernel):

(*results*) = *kernel*<<<grid,block>>>*(params).*

Здесь results - это список векторов, куда записывается результат выполнения ядра, а params - векторы, передаваемые ядру в качестве параметров. В угловых скобках указывается архитектура вычислительной системы для ядра: число блоков в решетке и число потоков в каждом блоке.

Потоки выполняются на cuda-ядрах – core (не путать с kernel – ядерными функциями), блоки - на мультипроцессорах. Решетка - абстракция для графического кластера. Блоки и решетки могут быть одномерными, двумерными и трехмерными. На каждый поток выделяется малый объем быстрой локальной памяти, на блок - небольшой объем быстрой общей памяти, на решетку выделяется медленная глобальная память очень большого объема. Кроме того, каждому ядру выделяется быстрая неизменяемая константная память, доступная каждому потоку в ядре.

Перед тем как описать програмную реализацию муравьиного алгоритма на графических процессорах, введем следующие обозначения:

v, v', **_v**, **-** векторы соответственно в глобальной, константной и общей памяти GPU; (tx,ty,tz) - индексы потока в блоке по соответствующим координатам; (bx,by,bz), (Dx,Dy,Dz) **-** индексы и размеры блока. Идентификатор потока в блоке определяется по формуле tid = tx+Dx*ty+Dz*tx*ty.

Например, для ядра *Hrc()*, рассчитывающего эвристическую информацию (его исходный код обсуждается далее), создается одномерная решетка из M двумерных блоков размерами Dx = Dy = N.

Ниже представлен исходный код программы, реализующей муравьиный алгоритм фолдинга белков на плоскости ACO-HP-PFP-2 (**A**nt **C**olony **O**ptimization for **H**idrophobic-**P**olar model **P**rotein **F**olding **P**roblem):

```
//Генерация случайных чисел для вычислительного эксперимента
(R) = Rnd(R,M*G*(L-1)); (I) = Rnd(I,M*G);
(τ) = IniTau<<<L-1,N>>>(τ, τ0);
for(g=0; Eg < Eb && g<=G-1; g++){ //цикл по всем колониям g из G
   (T) = IniIpt<<<M,1>>>(T,I,g);
   for(l = 0; l <= L-2; l++){ // получение сверток M-муравьев колонии g
      (da,la) = Actual<<<M,1>>>(da,la,I,l,g);
      (η) = Hrc<<<M,(N,N)>>>(η,T,la,da);
      (P) = Prb<<<M,N>>>(P,η,τ,la,α,β);
      (E,T,Y) = Stp<<<M,1>>>(P,R,η,la,d,l,g);
      }//l
      // поиск наилучшей свертки в колонии g
      (Eb,Kb) = Best<<<1,M>>>(Eb,Kb,E,g);
      // построение лучшего тура
      (Tb) = BestT<<<1,L>>>(Tb,Kb,T,g);
      //испарение и депозит феромонов
      (τ) = EvpTau<<<L-1, N>>>(τ,ρ);
      (τ) = DpsTau<<<M,L-1>>>(τ,Y,E,Eb,g);
// глобальное обновление феромонов лучшим туром
      (τ) = DpsTaubyBest<<<1,L-1>>>(τ,Y,Kb,g);
//обнуление энергий и туров перед запуском g+1-й колонии
      (Y,E,T) = IniAnts<<<M,L>>>(T,Y,E,0);
}//g
//Поиск наилучшего муравья среди всех поколений.
(G)=GlbBest<<<1,G>>>(G,Eb);
(G)=GlbBestT<<<1,L>>>(G, Tb);
```

Входными данными для программы являются: числа колоний муравьев - G и муравьев в колонии - M; число направлений сворачивания - N; коэффициенты α и β, учитывающие влияние феромонов и следов соответственно; коэффициент испарения следов - ρ и начальные значения матрицы феромонов - τ_0, а также вектора для: хранения последовательности аминокислот - Q, в которой «0» соответствует «P», а «1» - «H»; эвристики - **η**; феромонов - **τ**; вероятностей и случайных величин **P,R**; точек инициации сворачивания - **I**; тура муравья, представленного последовательностью узлов и направлений – **T,Y**; приращений в названиях узлов - **ΔT' = (**1, 2L, -1, -2L); энергий - **E**; актуальных на шаге l аминокислот и направлений сворачивания белков - $\mathbf{l_a}$, $\mathbf{d_a}$; глобальных значений энергии и свертки белковой последовательности - **G**; лучших муравьев - $\mathbf{K_b}$; их энергий - $\mathbf{E_b}$ и туров - $\mathbf{T_b}$ для каждого поколения.

Процесс создания ядер алгоритма рассмотрим на примере ядра эвристической информации:

(**η**) = *Hrc*<<<M,(N,N)>>>(**η,T,l**$_a$**,d**$_a$).

Составим таблицу размером NxN, в которой столбцы и строки соответствуют четырем направлениям сворачивания белка.

Если выбор направления x увеличивает число гидрофобных контактов с аминокислотой в направлении y, то на пересечении (x,y) устанавливается «1», если нет, то «0». Например, при выборе x = 1 эвристика увеличивается на 2, так как текущая гидрофобная аминокислота становится соседом для двух других таких же гидрофобных аминокислот, обозначенных на рис.1 черным цветом.

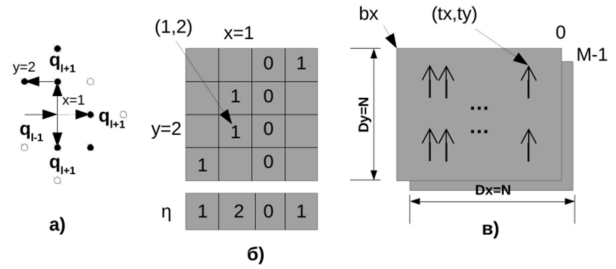

Рис. 1. Вычисление эвристической информации

Число таблиц должно совпадать с числом муравьев в колонии, и архитектура для этого ядра имеет вид <<<M,(N,N)>>>.

Исходный код ядра Hrc() следующий:

```
Hrc( η, T, l, d){
bs = Dx*bx; _η[ty]=1;
for(i=0;i<=L-1;i++) _T[i]=T[bx*L+i];
for(i=0; i<=L-1; i++){
   if(T[l[bx]]+ΔT'[tx]+ΔT'[ty]==T[bx*L+i])
         η[ty] +=Q'[i]*Q'[l_a[bx]+d_a[bx]]
}
for(i=0;i<=L-1;i++) { if(_T[l[bx]]+ΔT[ty] ==_T[i])
_η[ty]=0;}
η[bs+ty]=_η[ty];
}
```

Сначала общая память каждого блока решетки инициализируется начальными значениями эвристики (вектор **_η**) и частичными турами каждого муравья (вектор **_T**). Затем в каждом из (tx,ty) потоков проверяется, был ли посещен узел по заданному направлению ty. Если да, то **η**[ty] увеличивается на **Q'**[i]*$\mathbf{Q'}[\mathbf{l}_a[bx]+\mathbf{d}_a[bx]]$, при этом, если хотя бы одна из аминокислот окажется полярной (т.е. **Q'**[i] = 0 или $\mathbf{Q'}[\mathbf{l}_a[bx]+\mathbf{d}_a[bx]] = 0$), то приращение эвристики будет равно нулю. Здесь $\mathbf{l}_a[bx]$ и $\mathbf{d}_a[bx]$ — актуальные аминокислоты и направления для каждого муравья данной колонии, найденные ядром *Actual()* на предыдущем шаге алгоритма. Далее, если узел по направлению ty уже был посещен, то приращение эвристики также приравнивается к нулю. Ядро оканчивает работу записью эвристики из общей памяти в глобальную память gpu. Исходные коды остальных ядер приведены в таблице.

Исходные коды ядер алгоритма GPU-ACO-HP-PFP-2

IniTau()	**τ**[bx*Dx+tx] = τ_0
IniIpt()	i = L***I**[g*M+bx]; **T**[bx*L+i] = L*(2L+1)
Actual()	i=L***I**[g*M+bx]; $\mathbf{l}_a[bx]$ = (i+l ≤ L-2)? i+l : L-1-l; $\mathbf{d}_a[bx]$ = (i+l ≤ L-2)? 1 : -1
Hrc()	см. выше.
Prb()	bs =Dx*bx; y=tx; l_a=$\mathbf{l}_a[bx]$; _**η**[y]=**η**[bs+y]; _**P**[y]=_$\boldsymbol{\eta}^{\beta}$[y]*$\boldsymbol{\tau}^{\alpha}[l_a*N+y]$; for(s=0; s<=N-1; s++) Σ+=_**P**[s]; **P**[bs+y] = _**P**[y]/Σ;
Stp()	y=0; bs=Dx*bx; Σ = **P**[bs*N+y];
	while(Σ<**R**[g*M*(L-1)+l*M+bs]) { ++y; Σ+=**P**[bs*N+y]; } **T**[bs*L+l] = **T**[bs*L+l] +Δ**T**[y]; **Y**[bs*L+l]=y; **E**[bs]+=**η**[bs*N+y]
Best()	i=bx*Dx+tx; _**k**[tx]=tx; _**E**[tx]=**E**[i]; for(s=1;s<Dx;s*=2){ if(tx%(2*s)==0){ if(_**E**[tx]<_**E**[tx+s]){ _**E**[tx]=_**E**[tx+s]; _**k**[tx]=_**k**[tx+s];} }} if(tx==0){$\boldsymbol{K}_b$[g]=_**k**[0]; $\mathbf{E}_b$[g]=_**E**[0]}
BestT()	k_b = $\mathbf{K}_b$[g]; $\mathbf{T}_b$[g*L+tx]=**T**[k_b*Dx+tx]
EvpTau()	d=bx*Dx+tx; τ = **τ**[d] *= ρ; if(τ ≤ τ_{min}) **τ**[d]=τ_{min}
DpsTau()	d=bx*Dx+tx; if(d == **Y**[d]) { **τ**[tx*N+d]+= **E**[bx]/$\mathbf{E}_b$[g]; }
DpsTauby Best()	k_b = $\mathbf{K}_b$[g]; y=**Y**[k_b*Dx +tx]; **τ**[tx*N+y] += 1.0/Dx;
IniAnts()	i = bx*Dx+tx; **T**[i] = **Y**[i] = 0; **E**[bx] = 0;
GlbBest()	i = bx*Dx+tx; _**E**[tx] = $\mathbf{E}_b$[i]; _**g**[tx]=tx for(s=1;s<Dx;s*=2) {if(tx%(2*s)==0){ if(_**E**[tx] < _**E**[tx+s]) {_**E**[tx]=_**E**[tx+s]; _**g**[tx]=_**g**[tx+s];} }} if(tx==0) {**G**[L] =_g[0]; **G**[L+1] = _**E**[0]; }
GlbBestT()	b_g = **G**[L]; **G**[tx] = $\mathbf{T}_b[b_g*Dx+tx]$

Первый вызов ядра *Rnd()* генерирует все случайные числа для всех муравьев всех поколений и направлений сворачивания белка, второй вызов *Rnd()* - набор случайных чисел для выбора точек инициации сворачивания всех муравьев всех поколений. Ядро *IniTau()* инициализирует одновременно (L-1) - элементов матрицы феромонов **τ** начальными значениями τ_0. *IniIpt()* - инициализирует числами L*(2L+1) точки инициации сворачивания и первые узлы одновременно для M муравьев текущей колонии g**.** *Actual()* - выбирает текущие номера аминокислот и направления сворачивания (влево от точки инициации «-1», «+1» - вправо всех муравьев колонии g). Это нужно, чтобы облегчить вычисление эвристической информации.

Hrc() - расчет эвристической информации для M муравьев по всем направлениям (N,N) сворачивания по найденным ядром *Actual()* значениям l_a+ d_a .

Prb() - вычисление вероятностей выбора возможных направлений сворачивания одновременно для M муравьев. Принимает параметры α,β, вектор $\mathbf{l}_a$ - актуальных положений аминокислот для всех муравьев колонии g, указатель на матрицу феромонов **τ**. Актуальное положение аминокислоты - l_a для каждого муравья колонии g рассчитывается ядром *Actual()* по формуле: l_a = (i+l ≤ L-2)? i+l : L-1-l. Здесь i - положение точки инициации сворачивания, l - переменная цикла.

Результат ядра *Prb()* записывается в **P**.

Stp() - удлинение туров всех муравьев на одну позицию. Все предыдущие ядра подготавливали исходные данные для этого ядра. Вызванное в цикле

по l=0..L-1 оно выдает готовые свертки одновременно для M-муравьев колонии g. Результатом такого цикла будут туры **T** и значения энергий **E,** по которым можно судить, какой из муравьев выдал наилучшую свертку в поколении g.

Best() - поиск муравья (номер элемента вектора **E**) с наилучшим значением энергии (сам элемент) среди M-муравьев колонии g. Муравей и его энергия помещаются в g-е элементы векторов $\mathbf{K_b}$ и $\mathbf{E_b}$ соответственно.

BestT() - одновременное копирование L узлов тура лучшего муравья K_b из **T** в вектор $\mathbf{T_b}$ колонии g.

EvpTau() - испарение феромонов **τ** на величину ρ.

DpsTau() - локальное обновление феромонов турами **Y** муравьев колонии g; величина вклада определяется отношением энергий свертки каждого муравья (координаты вектора **E**) и лучшим значением энергии e_b среди g -колоний.

DpsTaubyBest() - глобальное обновление вектора феромонов **τ**, туром лучшего муравья (координатой вектора **Y**, определяемой по известному значению k_b) поколения g.

IniAnts() - обнуление энергий **E,** туров **T** и направлений сворачивания **Y** всех муравьев данной колонии.

GlbBest() - поиск наилучшего муравья и его энергии, которые записываются соответственно в **G**[L] и в **G**[L+1].

GlbBestT - одновременное копирование тура лучшего муравья в остальные L ячеек вектора **G**.

Итак, программа представляет собой последовательность ядер, выполняющих заданные операции над всеми муравьями одной колонии одновременно.

3. Обсуждение результатов

На рис.2 представлен результат выполнения потока, состоящего из одного задания, запущенного 17 декабря 2013 года в 14:13 по местному времени на рабочий узел: gpu.imit.uz с видеокартой Nvidia GTX-560Ti одноименного вычислительного ресурса, ресурсного центра «Института математики и информационных технологий (ИМИТ АН РУз), запущенного Базаровым Дмитрием Камильевичем – администратором виртуальной организации UZ-IMIT с веб-интерфейса доступа gUse-3.6.1 к ресурсам распределенной вычислительной системы на основе liferay-портала. Показана энергия E=7 и тур T лучшей свертки 176 поколения. Вычисления выполнены за 0.28 секунды.

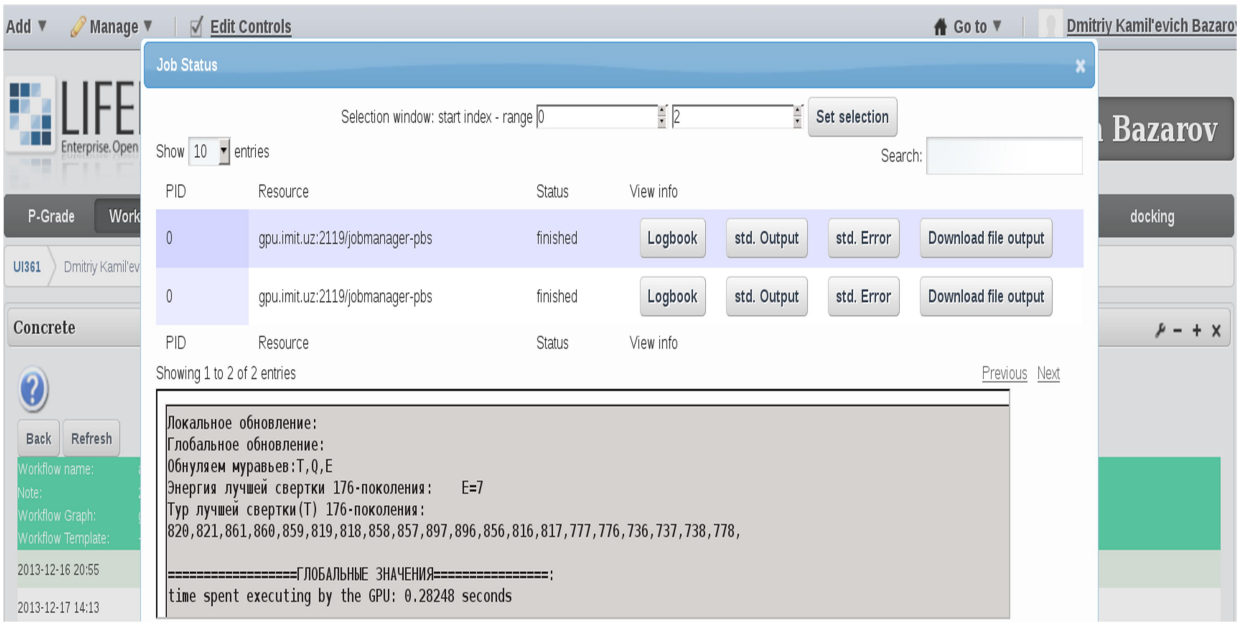

Рис. 2. Результат вычислительного эксперимента на веб-интерфейсе доступа к вычислительному ресурсу: gpu.imit.uz в видеокартой: GTX560Ti

Анализ результатов вычислительного эксперимента показывает, что данный способ распараллеливания подходит для моделирования процесса сворачивания крупных белков (от 50 аминокислотных остатков). Ведь даже последовательный перебор такого огромного числа колоний (1024 против 60) по сравнению с параллельной cpu-версией программы [14] для белка из 20 аминокислотных остатков осуществляется за доли секунды. Тем не менее, данный алгоритм не избавлен от недостатков, свойственных GPU-вычислениям: возникает ошибка запуска программы на видеокарте Nvidia GTX560Ti при входных данных: M > 32 и G > 1024; повторный запуск приводит к построению того же оптимального тура муравья с тем же значением энергии; 74% времени уходит на пересылки данных между устройством (GPU) и хостом (CPU). Несомненным достоинством графических процессоров по сравнению с дорогостоящим серверным оборудованием является их доступность.

Литература

[1] *Levinthal C*. How to Fold Graciously // Mossbauer Spectroscopy in Biological Systems: Proceedings of a meeting held at Allerton House, Monticello, Illinois. J.T.P. DeBrunner and E. Munck eds., University of Illinois Press. - № 41, 1969. - Pp. 22-24.

[2] *Mann, M., S. Will, and R. Backofen.* CPSP-tools - Exact and complete algorithms for high-throughput 3D lattice protein studies.Bmc Bioinformatics. - № 9(1), 2008. - 230 p.

[3] *Thachuk, C., A. Shmygelska, and H.H. Hoos,* A replica exchange Monte Carlo algorithm for protein folding in the HP model // Bmc Bioinformatics № 8, 2007. – 342 p.

[4] *Gordon S*. The self-avoiding walk: a brief survey // Proceednigs of the 33rd SPA Conference. Surveys in Stochastic Processes. Berlin: European Math. Society, 2010. - Pp. 181–199.

[5] *Shmygelska, A. and H.H. Hoos.* An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem // Bmc Bioinformatics. - № 6(1), 2005. – Pp. 30.

[6] *Thalheim T., Merkle D., Middendorf M.* Protein folding in the HP-model solved with a hybrid population based ACO Algorithm // IAENG International Journal of Computer Science. - № 35 (3), 2008. - Pp. 291-300.

[7] *Xiao-Min Hu, Jun Zhang, Yun Li.* Orthogonal methods based ant colony search for solving continious optimization problems // Journal of Computer Science and Technology. - № 23(1), 2008. - Pp. 2-18 (JCST_paper.pdf).

[8] *Stuzle T., Doringo M.* A short convergence proof for a class of ACO algorithms // IEEE Transactions on Evolutionary Computation. – № 6(4), 2002. - Pp. 358-365.

[9] *Carvelli L., Sebastiani G*. Some issues of aco algorithm convergence / Ant Colony Optimization - Methods and Applications, (Ed. A.Ostfeld) - InTech, 2011. - Pp. 39-52.

[10] *Гуляницкий Л. Рудык В.* Анализ алгоритмов прогнозирования третичной структуры протеина на базе метода оптимизации муравьиными колониями // Problems of Computer Intellectualization (Eds. V.Velichko, O.Voloshin, K.Markov). – Kiev-Sofia: V.M.Glushkov Institute of Cybernetics, ITHEA, 2012. – Pp. 152-159.

[11] *Chu D., Till M., Zomaya A.* Parallel ant colony optimization for 3D protein structure prediction using the HP lattice model // Proceedings 19th IEEE International Parallel and Distributed Processing Symposium. - № 07, 2005. - 193 p.

[12] *Delisle P., Krajecki M., Gravel M., Gagne C.* Parallel implementation of an ant colony optimization metaheuristic with OpenMP// Proceedings of the 3rd European Workshop on OpenMP (EWOMP'01), Barselona, 2001. - Pp. 8-12.

[13] *Wen-mei W. Hwu.* GPU computing gems emerald edition. - Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2011. – 886 p.

[14] *Бекмуратов Т.Ф., Мухамедиева Д.Т., Базаров Р., Ахмедов Д.Д.* Параллельный муравьиный алгоритм оптимизации // Узб. журнал «Проблемы информатики и энергетики». – Ташкент, 2014. - № 1-2. - С. 11-15.

[15] *Базаров Р.К.* Программный продукт: «Параллельная версия программы, реализующей муравьиный алгоритм сворачивания белоков на плоскости (ACO-HPPFP-2)» // АИС РУз. Свидетельство № DGU-02697. 26.12.2012 г.

[16] *Базаров Р.К.* Программный продукт: «Реализация муравьиного алгоритма для изучения фолдинга белков на плоскости методами программных агентов (Agent-ACOHPPFP-2)» // АИС РУз. Свидетельство № DGU-04104. 09.12.2016 г.

[17] *Базаров Р.К.* Программный продукт: «Реализация муравьиного алгоритма для изучения фолдинга белков на плоскости с помощью графических процессоров (GPU-ACO-HPPFP-2)» // АИС РУз. Свидетельство № DGU-03215. 08.07.2015 г.

[18] *Базаров Р.К.* Решение задачи фолдинга белков в распределенных вычислительных грид-системах // «Проблемы информационных и телекоммуникационных технологий»: Сб. докладов Республиканской научно-технической конференции. 10-11 марта 2016. - Ташкент, 2016. Ч.2. - С. 130-131.

[19] CUDA Zone [Электронный ресурс] / NVIDIA Corporation. – Режим доступа: https://developer.nvidia.com/cuda-zone.